UNIVERSITÉ DE MONTRÉAL

A DELAY-CONSTRAINED MIDDLEWARE ARCHITECTURE FOR DISSEMINATING INFORMATION IN WIRELESS SENSOR NETWORKS

JHON-FREDY LLANO-RUIZ

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

FÉVRIER 2011

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

A DELAY-CONSTRAINED MIDDLEWARE ARCHITECTURE FOR DISSEMINATING INFORMATION IN WIRELESS SENSOR NETWORKS

Présenté par : LLANO-RUIZ Jhon-Fredy

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

Mme. CHERIET Farida, Ph.D., présidente

M. QUINTERO Alejandro, Doct, membre et directeur de recherche

M. BEAUDRUN Ronald, Ph.D., membre et codirecteur de recherche

M. PIERRE Samuel, Ph.D., membre

DEDICATIONS

To Sofia, who lights up my life.

To my wife for her unconditional encouragement and unfailing support.

To my parents who always valued education and taught me hard work and perseverance.

ACKNOWLEDGEMENTS

To my director of research, Mr. Alejandro Quintero, and my co-director Mr. Ronald Beaudrun who provided me the means, the supervision and the advices to successfully complete this project.

To my dear friend Mr. Ebenezer Ebelle who was willing to share his knowledge and wisdom with me.

RÉSUMÉ

Les réseaux de capteurs sans fil (WSNs: Wireless Sensor Networks) constituent un domaine de recherche très actif et qui reçoit beaucoup d'attention de la part de la communauté scientifique. Ces réseaux offrent la possibilité de surveiller, de rassembler et de transmettre des données de l'environnement, en utilisant des nœuds appelés *capteurs*. Ces données sont généralement recueillies par une entité centrale qui les rend disponibles aux utilisateurs et à d'autres réseaux, comme Internet et les réseaux cellulaires. À cette fin, il existe des techniques, dites de diffusion, qui s'occupent de transférer les données à toutes les destinations. En particulier, le choix d'une technique de diffusion dépend de l'application utilisée. Par exemple, les applications à contraintes de délai imposent une contrainte de délai de bout-en-bout au processus de diffusion des données afin de coordonner des opérations de recherche et de sauvetage, et d'alerter les gens lors d'une opération d'urgence. Ce mémoire propose une architecture d'intergiciel qui sert de médiateur entre les applications à contraintes de délai et les techniques de diffusion de données. Une preuve de concept est faite pour montrer la faisabilité de l'architecture et l'efficacité d'un tel médiateur. L'analyse du prototype montre, entre autres, que le pourcentage de succès de transmission de l'intergiciel est largement supérieur à celui des protocoles individuels, tels que le service de messagerie (SMS: Short Message Service), le courriel et Twitter.

ABSTRACT

Wireless Sensor Networks (WSNs) have been gaining a lot of attention from the research community. They offer the possibility to monitor and collect information from the environment, using small sensor nodes. Generally, the information is later transmitted towards a centralized entity that serves as an interface to users, or to other networks (*e.g.*, cellular networks or Internet). Data dissemination techniques are in charge of the sending process to all destinations. The technique is chosen depending on the application. For instance, delay-constrained applications impose end-to-end delay constraints to coordinate rescue operations and warn people about critical events. This thesis proposes a middleware architecture that serves as a mediator between delay-constrained applications and data dissemination techniques. The architecture is intended to disseminate information from wireless sensor networks to Internet and cellular networks, considering end-to-end delay constraints. A proof of concept is done to validate its feasibility and effectiveness. More specifically, the results show that the percentage of success is much better using a middleware as a mediator than using individual protocols for data dissemination, such as Short Message Service (SMS), email and twitter.

CONDENSÉ EN FRANÇAIS

1. Introduction

Dans la dernière décennie, le monde a connu une très forte croissance en matière de développement et de déploiement des réseaux de capteurs (WSN) [1-3]. Le principal objectif d'un WSN est de recueillir l'information provenant de l'environnement, dans une région spécifique, afin de la rendre disponible aux différents usagers. Une fois que l'information est recueillie et traitée par le capteur, elle est retransmise vers une destination du même WSN, ou vers un autre réseau, comme Internet ou un réseau cellulaire [2-5]. Afin de transmettre efficacement les données sur le réseau, une technique de diffusion est nécessaire. La diffusion de données est un processus au cours duquel l'information est acheminée vers différentes destinations. Ce processus pourrait avoir une ou plusieurs contraintes de qualité de service (QdS) en fonction de l'application utilisée [1]. Par exemple, le délai de bout-en-bout, c'est-à-dire, le temps écoulé entre la source et la destination lors de la transmission d'un message, est un paramètre de qualité de service souvent utilisé [2]. Les applications à contraintes de délai imposent des contraintes de délais de bout-en-bout pour coordonner les opérations de sauvetage et informer le monde lors des opérations d'urgence [5]. Par conséquent, le délai entre chaque source et chaque destination doit être minutieusement surveillé. Pour ce faire, un médiateur (intergiciel) entre le réseau et les applications est nécessaire afin d'assurer le suivi du message. À l'aide d'un protocole de diffusion de données, l'intergiciel pourrait prendre des décisions à temps lorsque la contrainte de délai maximal de bout-en-bout n'est pas respectée. Ce mémoire propose un intergiciel à contraintes de délai pour la diffusion de l'information provenant d'un WSN vers d'autres réseaux.

1.1 Problématique

Les processus de diffusion des données p résentent deux problèmes majeurs. D'une part, le processus responsable d'acheminer l'information d'une source vers une destination est exécuté en utilisant un seul protocole ou une seule technique [2-5]. D'autre part, les propositions actuelles de diffusion de données sont élaborées pour être exécutées dans un environnement spécifique, comme un WSN ou Internet.

Les techniques actuelles de diffusion ne garantissent pas l'acheminement de l'information d'une source à une destination lorsque la diffusion des données se fait en utilisant plusieurs réseaux [3-5]. De plus, une seule technique est utilisée pour réaliser un tel processus, ce qui crée une dépendance entre le processus et le protocole choisi. Cette dépendance met à haut risque la qualité de service (QdS) souvent requise par les applications à contraintes de délai. À notre connaissance, il n'existe pas de médiateur pour coordonner le délai de bout-en-bout entre une source et une destination, indépendamment du réseau accédé et du protocole de diffusion.

1.2 Objectif

Le principal objectif de ce mémoire est de concevoir et de développer une architecture d'un intergiciel qui facilite la diffusion de l'information dans les réseaux de capteurs sans fil (WSN), en considérant des contraintes de qualité de service. Les objectifs spécifiques sont:

- Faire une revue de littérature sur les intergiciels et les techniques de diffusion de données dans les réseaux de capteurs sans fil, en considérant des algorithmes pour le délai de bout-en-bout et l'efficacité énergétique;
- Concevoir un intergiciel qui permet la diffusion des données à travers un réseau de capteurs sans fil et qui prend en considération le délai de bout-en-bout de la communication;
- Développer une validation de principe pour cet intergiciel qui envoie des messages entre les différents composants de l'architecture;
- Vérifier et valider l'architecture proposée, en utilisant une approche expérimentale qui permet d'évaluer la performance de cette architecture.

2. Revue de littérature

La revue de littérature a été faite sur deux points : les techniques de diffusion de données et les architectures existantes, c'est-à-dire les intergiciels et les cadriciels.

2.1 Diffusion des données

Cette section présente d'abord les paramètres importants liés au processus de diffusion des données. La deuxième partie utilise ces paramètres afin d'évaluer les protocoles présentés dans la revue de littérature.

2.1.1. Paramètres du processus

Voici les paramètres à considérer lors de l'analyse des protocoles de diffusion de données et de la conception de l'architecture :

- *Délai de bout-en-bout :* l'architecture doit garantir que l'information soit diffusée pendant un temps préétabli;
- *Efficacité énergétique :* la diffusion doit être faite de façon efficace; c'est-à-dire en optimisant la consommation d'énergie tout au long du processus;
- *Débit de transmission :* comme le délai est une contrainte importante, la solution proposée doit offrir un débit de transmission approprié;
- *Mécanismes de confirmation :* l'architecture doit permettre le suivi de chaque message envoyé par le système, en utilisant des mécanismes de confirmation. Elle doit pourvoir des stratégies réactives efficaces telles que, envoyer un message à travers un autre protocole de diffusion, lorsque le délai maximal est atteint;
- *Contrôle de la congestion :* le système doit avoir des mécanismes de contrôle de congestion qui minimisent les retransmissions, de manière à alléger les canaux du réseau;
- *Pourcentage de succès de transmission :* le rapport entre le nombre de paquets envoyés par un émetteur et celui reçu par un récepteur doit être proche de 1. En termes de pourcentage, il équivaut à 100%, ce qui signifie que le système est optimal.

2.1.2 Analyse des protocoles

Les protocoles présentés dans cette sous-section sont exclusivement conçus pour diffuser l'information. Le premier protocole étudié est CCBs (Concentric Circular Bands). Il est présenté par Ammari et Das [2]. Par ailleurs, on se base sur deux travaux qui se concentrent sur la comparaison des protocoles de diffusion de données [3, 4]. La première comparaison, faite par Zhang et Wang [4], décrit trois protocoles: DD (Direct Diffusion), TTDD (Two-Tier Data Dissemination) et GRAB (Gradient Broadcast). Dans la deuxième comparaison, Virmani et Jain [3] considèrent quatre protocoles: FDDDP (Forwarding Diffusion Data Dissemination), DDDP (Decentralized Data Dissemination), CBDDP (Credit Broadcast Data Dissemination) et EAGDDP (Energy Aware & Geographical Data Dissemination). La description des trois premiers protocoles est semblable à celle faite par Zhang et Wang [4]. Les auteurs intègrent aussi EAGDDP.

L'analyse, présentée au Tableau 1, a été faite comme un croisement entre les protocoles et les paramètres du système. L'évaluation considère les options suivantes :

- Paramètre fourni (Oui): le protocole prend en considération ce paramètre;
- Paramètre non fourni (Non) : le paramètre n'est pas fourni par le protocole;
- Paramètre ambigu (ND): rien n'indique si le paramètre est fourni ou non.

Table 1. Analyse des protocoles de diffusion de données

	Délai de bout-en-bout	Efficacité énergétique	Débit de transmission	Mécanismes de confirmation	Contrôle de la congestion
CCBs	Oui(*)	Oui(*)	Oui	Non	Non
DD	Non	Oui	Oui	ND	Oui
TTDD	Non	Oui	ND	Non	Oui
GRAB	Non	Non	ND	ND	ND
FDDDP	Non	Oui	Oui	ND	Oui
DDDP	Non	Oui	ND	Non	Oui
CBDDP	Non	Non	ND	ND	ND
EAGDD	Non	Oui	ND	ND	ND
(*) Le paramètre change en fonction de la configuration du protocole.					

Cette analyse permet de conclure qu'aucun protocole ne prend en compte tous les paramètres généraux. Par conséquent, l'intergiciel doit pourvoir des mécanismes pour pallier cette lacune.

2.2 Architectures

2.2.1 Aspects importants de l'architecture

L'intergiciel ne peut pas déléguer aux protocoles toutes les tâches reliées à la diffusion des données. Il doit s'adapter aux protocoles disponibles sans être dépendant de ceux-ci. Bulasubramanian et *al.* [11] présentent quatre aspects importants qui doivent être considérés pendant la conception d'un intergiciel. Ici, on a ajusté ces aspects selon les exigences de notre proposition:

- *Usage général :* même si on parle de diffusion de données, l'architecture ne doit pas faire d'hypothèses concernant les applications et les protocoles qui l'utilisent;
- *Transparence* : les applications n'ont pas besoin de connaître la façon dont l'intergiciel a été développé : l'utilisation des procédures doit se faire en utilisant des interfaces génériques;
- *Adaptabilité :* l'application doit s'adapter à certaines caractéristiques, comme le délai de bout-en-bout maximal et les destinations des messages;
- *Extensibilité :* la solution doit permettre d'étendre les fonctionnalités, en supportant de nouveaux protocoles de diffusion de données.

2.2.2 Analyse des intergiciels et des cadriciels

De nombreuses propositions ont été faites pour résoudre des problèmes courants dans les réseaux, comme la QdS [11-13], en utilisant des intergiciels. Cependant, ces intergiciels ne sont pas conçus pour s'exécuter dans les réseaux de capteurs. Parmi lesquels, on peut mentionner *Cygnus*, présenté par Balasubramanian *et al.* [11], *Chameleon* développé par Adam et Stadler [12], et pour terminer, MILCO présenté par Martin-Escalona *et al.* [13]. En contraste avec ce qui précède, les intergiciels proposés dans [7, 10] ont été conçus pour s'exécuter sur des technologies ad-hoc, comme un WSN ou un réseau véhiculaire (VANET : Vehicular Ad-hoc Network). Par exemple, Ribeiro *et al.* [7] présentent *SensorBus*. Un concept important mis en exergue dans cette application est l'inclusion des éditions. Un autre intergiciel implémenté sur VANETs est présenté par Delicato *et al.* [10]. Il est composé de trois perspectives : *device*, *network* et *application* pour gérer individuellement les configurations. Finalement, un cadriciel pour la diffusion des données dans les WSN est présenté par Saha and Matsumoto [5].

Tout comme lors de l'analyse des protocoles, on a fait un croisement entre les architectures (c'est-à-dire les Intergiciels et les cadriciels) et les aspects définis. L'évaluation est présentée au Tableau 2 et elle considère les options *Haut*, *Moyen*, *Bas*, *Aucun* ou *ND* (Non-déterminé).

Table 2. Analyse des intergiciels et cadriciels

	Usage général	Transparence	Adaptabilité	Extensibilité
Cygnus	Bas	Haut	Moyen	Moyen
Chameleon	Moyen	Haut	Haut	ND
VANET-intergiciel	Haut(*)	Haut	Bas	ND
MILCO	Bas	ND	Haut	Haut
SensorBus	Haut(*)	Haut	Moyen	ND
Cadriciel de gestion de catastrophes	Moyen	Aucun	Haut	Bas
(*) l'architecture prend en considération le concept d'éditions.				

Malgré certaines limitations de ces propositions, ces dernières ont défini plusieurs concepts d'importants qui seront considérés pendant la conception de notre architecture.

3. Architecture proposée

Le but principal de cette architecture est de satisfaire à la fois aux exigences du processus de diffusion et à celles de l'architecture présentée à la section 2. À cette fin, nous proposons une architecture d'intergiciel qui tient compte du délai de bout-en-bout pour la diffusion des données dans les réseaux de capteurs. Cette partie présentera d'abord un modèle mathématique qui permet de mieux comprendre les contraintes de délais imposées. Ensuite, l'architecture de référence sera expliquée.

3.1 Modèle mathématique

Afin de concevoir l'architecture, il est important de comprendre la modélisation mathématique du délai de bout-en-bout. La figure 1 illustre le processus d'acheminement de

l'information de A à F, en passant par plusieurs nœuds intermédiaires. À chaque segment, on enregistre un délai entre un émetteur et un récepteur.

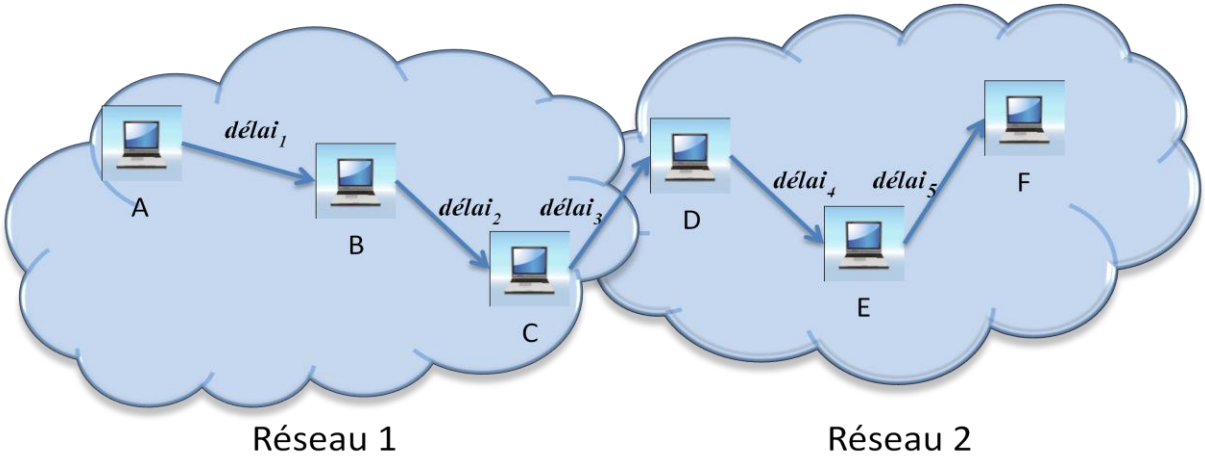

Figure 1. Délai de bout-en-bout entre les nœuds A et F

Dans ce cas, trois scénarios sont envisageables:

1) L'émetteur reçoit une réponse positive du récepteur dans une période de temps valide,

2) L'émetteur reçoit une réponse négative du récepteur dans une période de temps valide,

3) L'émetteur ne reçoit pas de réponse du récepteur.

Le délai varie d'un scenario à l'autre. Chaque délai est décri ci-dessous :

1) $délai_{ACK}$: Il est le temps écoulé entre l'envoie d'un message par l'émetteur et la réception d'une réponse positive, c'est-à-dire un ACK, dans une période de temps valide.

2) $délai_{NACK}$: Il est le temps écoulé entre l'envoie d'un message par l'émetteur et la réception d'une réponse négative, c'est-à-dire un NACK, dans une période de temps valide,

3) $délai_{sansreponse}$: Il est le temps écoulé entre l'envoie d'un message par l'émetteur et le moment auquel le processus de recherche d'un nouveau protocole de diffusion est terminé, étant donne que l'émetteur ne reçoit pas de réponse du récepteur, c'est-à-dire un NR.

Les équations (1), (2) et (3) sont les modèles qui décrivent chacun des délais:

$$délai_{ACK} = (T_i - T_0) + (T_{i+j} - T_i) \quad \textbf{(1)}$$

où T_0 = Temps d'initialisation de l'exécution, T_i = Temps où le protocole de diffusion des données est appelé, T_{i+j} = Temps de réception de la réponse du protocole.

$$délai_{NACK} = (T_i - T_0) + (T_{i+j} - T_i) + (T_{i+j+k} - T_{i+j}) \quad \textbf{(2)}$$

où T_{i+j+k}= Temps où le processus de recherche d'un nouveau protocole de diffusion est terminé.

$$délai_{sansreponse} = (T_i - T_0) + \alpha_j + (T_{i+\alpha_j+k} - T_{i+\alpha_j}) \quad \textbf{(3)}$$

où α_j= Temps maximal pris par un protocole pour finaliser une tâche, $T_{i+\alpha_j+k}$= Temps où le processus de recherche d'un nouveau protocole de diffusion est terminé lorsque le protocole utilisé précédemment a pris le temps maximal d'exécution (c.-à-d. il n'y a pas eu de réponse), $T_{i+\alpha_j}$= Temps où l'intergiciel commence le processus de recherche d'un nouveau protocole de diffusion car le protocole précédent n'a pas répondu.

Le $délai_S$ fait référence au délai enregistré par chaque paire de nœuds : par exemple, entre les nœuds A et B de la Figure 1. Ce délai correspond à la somme de tous les $délai_{NACK}$, les $délai_{sansreponse}$ et un $délai_{ACK}$. Finalement, l'équation (4) représente le délai de bout-en-bout, c'est-à-dire la somme des $délai_S$ pour tous les nœuds entre la source et la destination (entre A et F de la Figure 1).

$$délai_S = (délai_{ACK}) + \sum_{i=1}^{w} (délai_{NACK})_i + \sum_{j=1}^{x} (délai_{sansreponse})_j$$

où *w* est le nombre de transmissions avec une réponse négative et *x* est le nombre de transmissions sans réponse.

$$délai_{bout-en-bout} = \sum_{S=1}^{N-1} délai_S \quad \textbf{(4)}$$

où *N*= Nombre de nœuds inclus dans le processus.

3.2 Architecture de référence

L'architecture de référence proposée comprendra trois couches et trois plans de services (Figure 2). D'un côté, les couches sont : *interface*, *règles de gestion* et *service de traitement de*

données. De l'autre côté, les plans de services agissant sur les trois couches sont : *QoS*, *Confirmation* et *Rapidité de diffusion*.

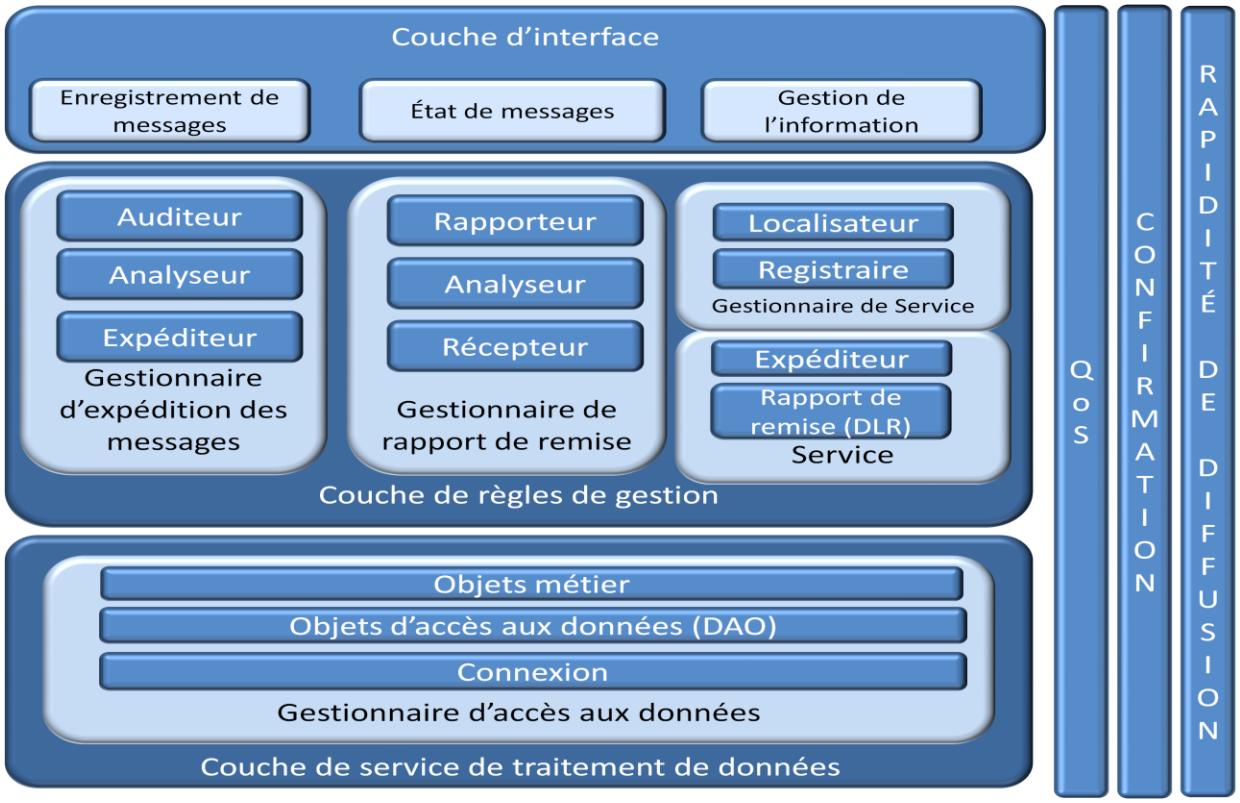

Figure 2. Architecture de référence

La couche d'interface offre un service d'accès standard qui donne aux émetteurs une façon unique d'enregistrer de nouveaux messages, d'obtenir l'état des messages envoyés et de gérer les données utilisées dans le processus. La couche de règles de gestion offre les fonctionnalités de base du système. Sa principale responsabilité est de garantir le processus d'expédition des messages et le processus de rapport de remise. Elle gère aussi la communication avec les différents protocoles de diffusion des données. Finalement, la couche de service de traitement de données a pour but de donner l'accès aux données sauvegardées par l'intergiciel. Les plans des services sont orientés de manière à offrir le délai de bout-en-bout, le processus de confirmation et la rapidité de diffusion de l'information.

L'intergiciel est conçu pour être déployé sur trois équipements différents: les capteurs sans fil, la passerelle et la station de base. En utilisant le concept d'édition, une variété de l'intergiciel est exécutée sur chaque équipement en particulier. Ainsi, trois éditions sont définies dans cette architecture : SN-Edition (édition du capteur), GW-Edition (édition de la passerelle) et BS-Edition (édition de la station de base).

4. Validation de principe et résultats

Après avoir présenté l'architecture, nous utilisons une validation de principe pour évaluer la faisabilité du système. Dans le scénario sélectionné pour la validation, les capteurs sans fil envoient des informations sur le degré de luminosité d'une pièce vers les utilisateurs de réseaux cellulaires et d'Internet, dès que ce degré tombe en-dessous d'un seuil préétabli. Pour le développement du prototype, on considère un sous-ensemble de l'architecture globale. Ce sous-ensemble inclut l'édition de la station de base (BS-Edition), qui prendra en considération toutes les fonctionnalités de base du système. Seules les fonctionnalités considérées critiques pour la validation ont été développées.

Avant l'exécution du programme, nous réexaminons le modèle mathématique du délai de bout-en-bout, étant donné que les délais de l'intergiciel et de recherche d'un nouveau protocole de diffusion sont négligeables. Les équations résultantes, délai avec ACK (5), délai avec NACK (6) et délai sans réponse (7), sont:

$$délai_{ACK} = (T_{i+j} - T_i) \quad \textbf{(5)}$$

$$délai_{NACK} = (T_{i+j} - T_i) \quad \textbf{(6)}$$

$$délai_{sansreponse} = \alpha_j \quad \textbf{(7)}$$

Pour faire la validation des résultats, 20 expériences ont été effectuées. Pour chaque expérience, 400 messages ont été envoyés aux utilisateurs et trois protocoles ont été utilisés : le SMS, le courriel et Twitter. Chaque délai est enregistré séparément. Sur la base de ces informations, deux analyses sont effectuées : l'analyse des messages individuels et l'analyse des expériences. Dans la première analyse, 12 messages sont sélectionnés de façon aléatoire. On peut

alors apprécier les décisions de l'intergiciel sur la base du délai de bout-en-bout et de l'état de la confirmation enregistrés par chaque protocole. Dans la deuxième analyse, trois perspectives sont à considérer : le délai maximal de bout-en-bout, le délai moyen de bout-en-bout et le pourcentage de succès de la transmission. Le traitement du premier cas garantit que le délai ne dépasse jamais un certain seuil. Quant au délai moyen, il montre comment l'efficacité peut être améliorée. Pour terminer, le pourcentage de succès de la transmission est aussi analysé, ce qui révèle un pourcentage de l'intergiciel proche de 98%, largement supérieur au succès des ressources individuelles, telles que le SMS (78%), le courriel (79%) et Twitter (61%).

5. Conclusion et travaux futurs

5.1 Conclusion

L'analyse des résultats montre que les exigences de diffusion relatives aux données du système sont satisfaites. Plus particulièrement, le délai de bout-en-bout est satisfait en utilisant une période de temps maximal pour chaque protocole. De plus, le pourcentage de succès de la transmission est satisfait grâce à l'inclusion de plusieurs ressources pour la diffusion des données. Nous pouvons aussi conclure que l'efficacité énergétique, le débit de transmission, les mécanismes de confirmation et le contrôle de congestion sont aussi satisfaits. L'efficacité énergétique est obtenue à partir du fait que l'architecture peut prendre la décision de diffuser ou non certaines informations. De plus, en utilisant le composant pour le *rapport de remise*, l'architecture fait le dépistage de chaque message en tout temps, offrant ainsi des mécanismes de confirmation basés sur l'état de chaque message, c'est-à-dire basés sur des ACK, NACK et NR. Ce composant donne aussi la possibilité au système d'utiliser plusieurs ressources en même temps, afin de faire de la notification un processus efficace, ce qui garantit le requis de débit de transmission. En outre, le composant DLR aide au contrôle de congestion car l'état de chaque message est connu en avance, en évitant ainsi les retransmissions.

Les exigences concernant l'architecture sont aussi satisfaites. L'architecture est considérée d'usage général car elle est totalement indépendante des applications et des protocoles de diffusion de données. Par ailleurs, l'exigence de transparence est satisfaite grâce à l'approche orientée-interface sur laquelle le système a été désigné. De plus, afin de faire que l'intergiciel ait adaptabilité certains paramètres, tels que le délai maximal, sont adaptables. Finalement,

l’architecture est extensible car le design considère l’inclusion future de nouvelles fonctionnalités telles que protocoles de diffusion de données.

5.2 Limitations du travail

Le mémoire comporte quelques limitations. Premièrement, l’intergiciel a une forte dépendance par rapport au protocole de diffusion utilisé. Ainsi, si le protocole ne permet pas de récupérer l’information concernant l’état de chaque message, il ne pourra pas être utilisé dans l’architecture. Une autre limitation est la dépendance de l’architecture de la passerelle qui permet la communication et la diffusion d’informations vers les réseaux cellulaires. En effet, si la passerelle est absente, la diffusion ne peut être garantie. Finalement, on suppose que chaque dispositif recevant un message aura le moyen d’envoyer une réponse, en confirmant une réception positive ou négative. Si cette fonctionnalité est absente, la remise ne peut être assurée.

5.3 Travaux futurs

Les travaux futurs peuvent porter sur plusieurs points. Une première proposition est la finalisation du développement des trois éditions, c'est-à-dire l’édition du capteur (SN-Edition), l’édition de la passerelle (GW-Edition) et l’édition de la station de base (BS-Edition), afin de pouvoir évaluer la totalité du système. Un autre travail proposé est la réalisation d’un test, en utilisant des protocoles réels au lieu d’une approche probabiliste qui simule les délais et les confirmations. D’autres travaux consisteraient à étendre l’intergiciel au-delà des WSNs. Pour ce faire, il faudrait concevoir de nouvelles éditions qui s’exécutent sur les dispositifs des utilisateurs, comme des ordinateurs portables, des PCs et téléphones intelligents, et des téléphones cellulaires. Également et à propos de l’attention qu’a reçu le domaine véhiculaire dans la dernière décennie, nous proposons de développer et de déployer une édition pour les réseaux véhiculaires (VANETs). Cependant, il faudrait faire une étude approfondie concernant ses architectures, comme WAVE et ses protocoles de diffusion de données.

INDEX

INDEX OF TABLES

INDEX OF FIGURES

LIST OF ACRONYSMS AND ABREVIATIONS

4G	Fourth Generation
ACK	Positive Acknowledgement
AE	Application Engineering
API	Application Program Interfaces
ARS	Ad hoc Relay Stations
BO	Business Object
BR	Business Rules
BS	Base Station
BS-Edition	Base Station Edition
CBDDP	Credit Broadcast Data Dissemination
CCBs	Concentric Circular Band
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DAO	Data Access Object
DD	Direct Diffusion
DDDP	Decentralized Data Dissemination
DE	Domain Environment
DLR	Delivery Report
DM	Delivery Manager
DN	Dissemination Node
DoI	Degree of Interest
DS	Data Services
EAGDDP	Energy Aware & Geographical Data Dissemination

E/R	Entity/Relation
FDDDP	Forwarding Diffusion Data Dissemination
GW	Gateway
GW-Edition	Gateway Edition
GRAB	Gradient Broadcast
HTML	Hyper-Text Markup Language
I/O	Input/Output
IMT	International Mobile Telecommunications
IX	Interface
LEPSM	Link Estimation and Parent Selection Method
MO	Message Originator
MSM	Message Sender Manager
MT	Message Terminator
NACK	Negative Acknowledgment
NR	No Response
OS	Operating System
PC	Personal Computer
PL	Programming Language
PLA	Product Line Architecture
QoS	Quality of Service
SENDROM	Sensor networks for disaster relief operations management
S	Segment
SG	Security Group
SM	Service Manager

SMS	Short Message Service
SN	Sensor Node
SN-Edition	Sensor Node Edition
SPL	Software Product Line
SPLE	Software Product Line Engineering
TTDD	Two-Tier Data Dissemination
UG	User Group
UWSN	Under Water Sensor Networks
VANET	Vehicular Ad-hoc Network
WAVE	Wireless Access in Vehicular Environments
WIMAX	Worldwide Interoperability for Microwave Access
WSN	Wireless Sensor Network
XML	eXtended Markup Language

LIST OF APPENDICES

Chapter 1 INTRODUCTION

In the last decade, the world has witnessed a tremendous increase in development and deployment of Wireless Sensor Networks (WSNs) [1-3]. The main goal of a WSN is to gather environmental information in a specific region and make it available to users. For this purpose, it uses a sensor node, *i.e.*, a device that measures environmental variables, such as light, temperature, humidity and barometric pressure among others [1]. Sensor nodes have three main responsibilities: sensing, measuring and gathering information from the environment [1]. Once the information is sensed and processed inside the node, it is transmitted to a destination which might be another device either inside the WSN or in another network (*e.g.*, Internet or Cellular Network) [2-5]. In order to effectively transmit data along the network, a data dissemination technique is required. Data dissemination is the process where information is transported towards different destinations. Depending on the application, this process might have one or more Quality of Service (QoS) constraints [1]. For instance, the end-to-end delay, *i.e.*, the time elapsed between the source and the destination to transmit a message, is a QoS parameter often used [2]. Delay-constrained applications impose end-to-end delay constraints to prevent accidents, coordinate rescue operations and warn people about critical events [5]. Accordingly, in such applications, the delay from each source to each destination should be monitored. For this purpose, a mediator (middleware) between the network and the applications is required to offer tracking capabilities of the disseminated information. Using a data dissemination protocol, this middleware could take on-time decisions when a maximum end-to-end delay constraint is exceeded. This thesis proposes a delay-constrained middleware for disseminating information from a WSN to other networks.

The rest of this chapter is organized as follows. The first section introduces basic concepts necessary for understanding the thesis. Then, the aspects of the problem are presented. Later on, the research objectives are listed. The organization of the thesis is exposed in the final section.

1.1 Definitions and basic concepts

As can be seen in Figure 1-1, the main entity of a WSN is a sensor node (SN). SNs, also known as multi-purpose nodes, are intended to measure several environmental variables during a certain time, such as light, temperature, humidity and barometric pressure [1]. They have three

main responsibilities: sensing, measuring and gathering information from the environment. Depending on the application requirements, once the information is sensed and processed, the SN sends this information to a destination by using a data-dissemination technique.

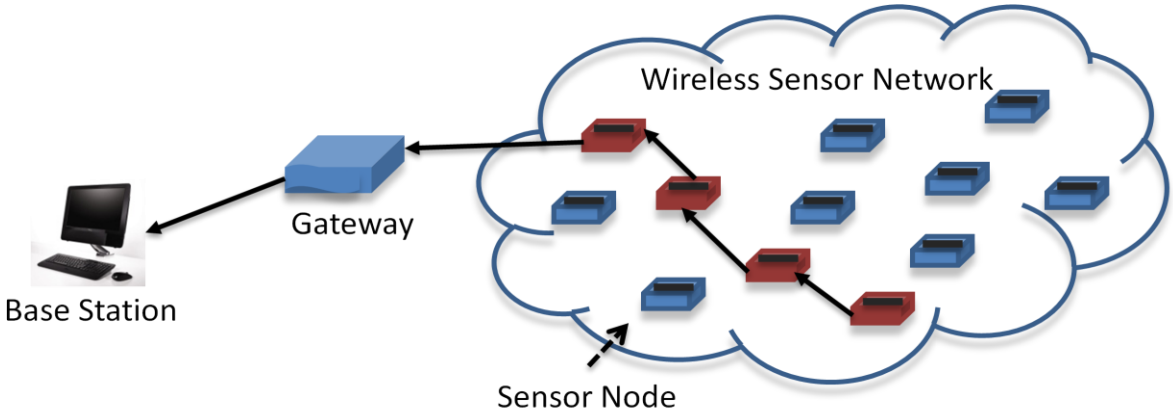

Figure 1-1 Example of a Wireless Sensor Network

Another important component of a WSN is the Base Station (BS). Since sensors are normally battery-constrained and equipped with low system capabilities, they need to transfer their collected data to a long-life device. Laptops, Personal Computers (PCs), handhelds and access points to a fixed infrastructure are examples of physical devices used as BS. To make the communication possible between SNs and BSs, a node called gateway (GW) is set in between, acting as a bridge. It has higher system, transmission and power capabilities than a generic node. The exchange of information between these three types of devices is done through a data-dissemination technique [2-5]. The selected data dissemination technique has to consider the use of different paths while satisfying different requirements.

Data dissemination process may vary depending on the architecture and network topology. Therefore, several variables are taken into consideration: architectural variables (the gateway is static or mobile) [6], routing variables (single or multiple hops) [2] and processing variables (data aggregation techniques are present or not) [1]. When defining or selecting data dissemination algorithms, all these factors are considered due to their impact on the process behaviour (*e.g.*, end-to-end delay) [6]. Particularly, when working on delay-constrained environments, the end-to-end delay parameter should be carefully analyzed. As can be seen in Figure 1-2, such delay can represent a single or multiple hops. For instance, to convey the

information from A to F, the end-to-end delay is the summation of each individual delay (*i.e.*, $delay_1, delay_2, delay_3, delay_4, delay_5$).

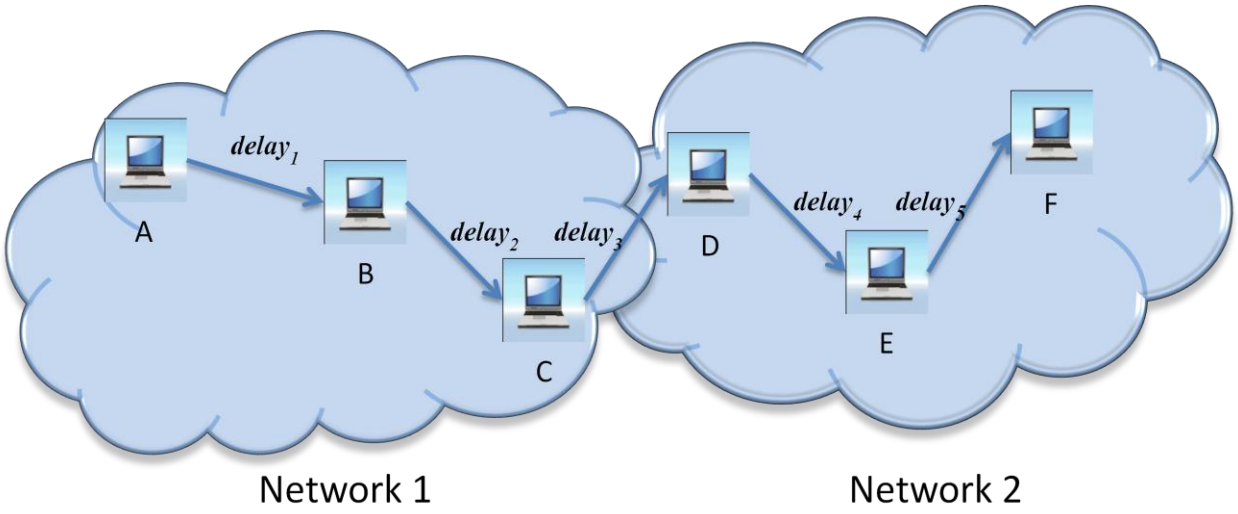

Figure 1-2 End-to-end delay between nodes A and F

Since delay-constrained applications and data dissemination protocols should not be connected directly, there is a mediator in between, *i.e.*, a middleware (intermediary level intended to connect two components). The communication and coordination among the components is usually done through messages [7]. It avoids applications to access directly the low-level features, such as protocols and hardware. This offers a high level of abstraction, normally exposed as interfaces. For this purpose, the middleware should be deployed in each single device involved in the data dissemination process. For example, nodes A to F in Figure 1-2 should have the middleware deployed.

1.2 Aspects of the problem

Data dissemination processes have two major issues to be considered. On one hand, the process responsible for conveying the information from a source to a destination is executed using a single protocol or technique [2-5]. Even if some of the protocol proposals offer certain QoS parameters, such as end-to-end delay [2, 8], the applications still have to rely on a single option to convey information when using these techniques. On the other hand, current data dissemination proposals are designed to be executed on a specific environment (*e.g.*, a WSN or Internet). For instance, techniques such as [2-5] are especially designed to be performed on WSN.

These characteristics result in serious drawbacks when disseminating information among several networks.

When data dissemination is required along more than one network, the existing techniques do not guarantee to convey information from a source to a destination due to its inherent nature [3-5]. Even worst, the dissemination depends hardly on one single technique to perform such process, which creates a complete dependency on the dissemination protocol being used. These shortcomings could lead the propagation of information to be seriously compromised. In hazardous events, for instance, the dissemination should not depend on a single protocol or means, as stated by AT&T [9]. Indeed, protocols deal with uncontrollable variables that make it impossible for them to assure a specific level of QoS or reliability [2]. As a consequence, a lack of certainty about the status of the messages in the network arises. Therefore, the high dependency on a single data dissemination technique puts at serious risk the QoS often required by delay-constrained applications. Up to our knowledge, there is not any mediator proposal to coordinate the end-to-end delay between a source and a destination independently from the accessed network and dissemination protocol.

1.3 **Research objectives**

Our main goal is to design and develop a middleware architecture that enables the dissemination of information in WSN considering QoS constraints. More precisely, this research project has the following goals:

- Study and analyze the state of the art in middlewares and information dissemination in WSN considering end-to-end delay communication and energy efficient algorithms existing in these networks.
- Design a middleware that enables data dissemination through WSN by considering end-to-end delay communication as QoS parameter.
- Develop a proof of concept of the middleware that sends messages among the different components of the architecture.
- Verify the proposed solution by using an experimental approach that let validate the architecture performance.

1.4 **Outline**

The rest of this dissertation is organized as follows. Chapter II presents the state of the art regarding WSN, standards, system, network services, protocols, architectures and data dissemination solutions. Chapter III presents the proposed architecture. Chapter IV is committed to the proof of concept implementation and experimental verification of the proposed architecture. And finally, we present the conclusions and future work.

Chapter 2 STATE OF THE ART

It is widely accepted the impact data dissemination processes have on human lives [5, 9]. Particularly, when dissemination is performed in challenging environments, *e.g.*, WSNs, the techniques used for such a purpose become tremendously critical. Fortunately, research community has been hardly working on this subject in recent years [3, 4]. This chapter, thus, is intended to know the state of the art in some important matters, such as WSNs, data dissemination protocols and existing architectures (*i.e.*, middleware or framework), due to their direct impact on the development of this thesis.

The rest of this chapter is organized as follows. The first section initially gives a brief description of the logical architecture of a WSN. The second and the third sections present the state of the art from two perspectives: 1) data dissemination techniques and 2) generic middlewares and frameworks. Moreover, some the global parameters are defined in each perspective in order to evaluate either a data dissemination technique or an architecture. These parameters are then integrated in final section to show a complete view of the middleware-architecture to be built.

2.1 Logical architecture in WSNs

According to [1], WSN main goal is to gather environmental information from a specific region where sensors are deployed. It is generally comprised from a few tens to thousands of sensor nodes and from one to several base stations deployed in a geographic region. Two types of deployments can be found in these types of networks: *structured* and *unstructured*. An *unstructured* network, also known as ad-hoc, is where nodes are deployed randomly (*e.g.*, underwater measures, high temperatures and extreme pressures). In this type of network, minimal or no administration is done to the network. Therefore, diagnosing failures is quite difficult. In contrast, a *structured* deployment distributes the nodes among the field in a pre-known-basis (*e.g.*, building surveillance, car/bus tracking and patient monitoring); which reduces maintenance cost, the number of nodes in the network and failure-detection efforts. Even though, structured sensor networks are easier to manage, unstructured networks give the advantage to be deployed when human intervention is not possible. The decision to choose one or another is imposed mainly by application requirements and constraints. Several constraints need to be considered

when analyzing WSNs. Energy restrictions, short communication range, low bandwidth and limited system capabilities are some of these restrictions. Unlike, other similar networks such as Vehicular Ad-hoc Networks (VANETs) [10], WSNs need to pay special attention to energy consumption algorithms since they are energy constrained.

There are three main aspects that need to be addressed when designing WSN solutions according to [1]. Firstly, the *system*; associated to platforms, operating systems and storage. Secondly, the *communication protocols*; acting as a bridge between the application and the sensors in the network. It includes aspects, such as data dissemination and QoS. And finally, the *services*, devoted to improve system performance and network efficiency. Since the *communication protocols* play an important role in WSNs, it is important to analyze their architecture. This technology defines a protocol stack that consists of five layers. From top to bottom the defined layers are: application, transport, network, data-link and physical as stated by Yick *et al.* [1]. Each sensor uses this stack to communicate with each other, the gateway and the base station. Figure 2-1 presents the WSN communication model. As can be seen, the application layer is the upper layer in the stack. This research focuses on proposing a middleware to disseminate information in WSNs at this level. Nonetheless, it is important to know the state of the art regarding data dissemination techniques in the lower layers since they are the main support to convey the information towards the destinations.

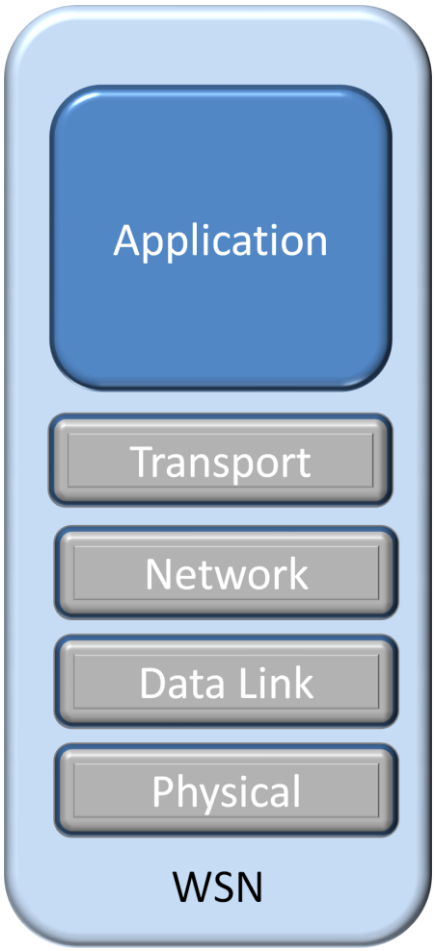

Figure 2-1 WSN communication model

2.2 Data dissemination

Data dissemination is the process in charge of transporting information towards different destinations. The first part of this section defines some important parameters related to this process. Later on, proposed data dissemination protocols are reviewed. Finally, in the light of the defined parameters an analysis is made to each protocol.

2.2.1 Data dissemination parameters

Before analyzing in further detail current proposals for data dissemination in WSNs, some important parameters are defined. They express the basic needs for an architecture that will enable the dissemination of information in WSN considering QoS constraints.

- *End-to-End delay*. The architecture should offer a way to guarantee the information is disseminated in a pre-established period of time. If during this time the information has not reached its destination, the source should have a mechanism to forward the information by using another data dissemination protocol.
- *Energy efficiency*. Data dissemination should be done in an efficient way. Efficiency is defined as the optimization of the energy consumption along the whole process.
- *Transmission rate.* Since information needs to be disseminated with delay-constraints, the solution should offer a scheme where an appropriated rate is assured.
- *Confirmation mechanisms.* Each message sent to a destination should be tracked down. It means that positives and negatives feedbacks are mandatory to the architecture in order to provide efficient reactive strategies, *i.e.*, send the message through another data dissemination protocol when a maximum delay is exceeded. A reactive strategy is triggered when a negative acknowledge or no response is obtained from the data dissemination protocol.
- *Congestion Control*. The system should be equipped with congestion control capabilities in order to reduce retransmissions, which in turn, alleviate the network channels since they are not busy with unnecessary information. As a result, the energy consumption is also reduced as sensors only transmit when required.
- *Percentage of success*. The relation between the number of messages sent by a source and the number of messages received by a destination should be closer to 1, It means that the system is considered to be completely optimal when the all the messages were correctly received by the destination minimizing retransmissions. In this case the percentage of success is said to be 100%.

It is important to notice that depending on the type of application these parameters might be fully considered. This is the case of delay-constrained applications due to their criticism as they are intended in some cases to save people's lives.

2.2.2 Protocols

There is a wide variety of events that might affect QoS. For instance, when a sensor runs out of battery, all communications that rely on that path are affected, producing a cascade effect [1]. Therefore, nodes are forced to find alternate routes which consequently could not be the most optimal communication. Consequently, these changes could affect other parameters, such as end-to-end delay within the whole network. Solutions to treat this problematic have been proposed by researchers in the last decade. Efficient wireless communication, predefined sensor placement, security, efficient storage, aggregation and compression techniques are among the methods that can be used to overcome these shortcomings. Particularly, regarding efficient wireless communication, data dissemination protocols have been proposed [2-4]. The protocols presented in this sub-section are exclusively intended to disseminate information. Each of them will be analyzed in the light of the parameters previously defined (*e.g.,* end-to-end delay, energy efficiency, transmission rate, confirmation mechanisms, congestion control). It will serve as a guide to the right choice of these protocols, since it is considered as a critical point that impacts the performance and the results when conveying information to a user.

The first data dissemination protocol studied is presented by Ammari and Das [2]. They developed a rigorously mathematical proposition. It trades-off energy consumption and end-to-end delay in WSNs. They use a concept called Concentric Circular Bands (CCBs) to decompose the transmission range of sensors based on the minimum distance between two sensors. It enables sensors to express its Degree of Interest (DoI). This value is used to minimize either energy or delay or both. They propose a data dissemination protocol using these concepts. This protocol meets application requirements such as rapid transmission-rate or energy efficiency. It makes five important assumptions. 1) The deployment is uniformly distributed 2) only one gateway is used 3) transmission range is equal for all the sources. Gateway transmission range is larger 4) the nodes know in advance their own location and make a location advertisement at the beginning 5) sensors inform their neighbors about their energy status.

There are two other works that focus in the comparison of data dissemination protocols [3, 4]. These insights help to identify the strengths and the weaknesses of each proposition. The rest of this section is dedicated to initially describe the protocols and then after, such contributions are analyzed in the light of the data dissemination parameters. The first comparison

made by Zhang and Wang [4] describes three data dissemination protocols in WSNs: Direct Diffusion (DD), Two-Tier Data Dissemination (TTDD) and Gradient Broadcast (GRAB). Firstly, DD is a data-centric communication protocol. It uses attribute-value pairs to represent the information generated by the source. When a destination needs information it broadcasts manifesting its interest in a periodic way to get the data. Gradients are established within neighbors and it contains the data transmission rate and the direction of the information. Once a source identifies an interested destination it sends exploratory packets to it. When the destination receives this information it chooses a neighbor who will be responsible to address the data towards it. Secondly, TTDD is a dissemination protocol that divides the network topology into cells. Whether to forward data or not is determined by the location of the sensor. Only those inside the cell can do it. The collection of the cells is seen as a grid and its construction is a process done by the destination. Additionally, it defines the forwarders that are known as Dissemination Nodes (DNs). It defines a concept called two-tier, which refers to two cells; one is the destination location and the other one is the DN at cell boundaries. To convey the information the destination floods a query in its cell, the closest DN will receive that request, then all the communication is done through the DNs until the source is reached. It can also be terminated when a DN that contains the corresponding data is attained. Finally, GRAB supposes that the cost for a node to reach a destination is infinity. When a destination node starts, it broadcasts an advertisement message with the initial cost. All the nodes in between that receive the message calculate the cost of that message to arrive. It allows all nodes to know the minimum cost to attain the destination when the setup process is finished.

In the second comparison, a quite similar and more recent study is done by Virmani and Jain [3]. The study considers four propositions: 1) Forwarding Diffusion Data Dissemination (FDDDP), 2) Decentralized Data Dissemination (DDDP), 3) Credit Broadcast Data Dissemination (CBDDP) and 4) Energy Aware & Geographical Data Dissemination (EAGDDP). Accordingly, the description of the first three protocols in this study is almost the same compared to the previously ones described by Zhang and Wang [4]. Therefore, it is not consider necessary to detail them again. However, more nodes in the simulations were included. Additionally, a protocol called EAGDDP was also included. EAGDDP considers residual energy and efficient query dissemination. Instead of using flooding techniques the packets are addressed to a specific destination. Each node knows beforehand its own position.

2.2.3 Analysis to protocols

This subsection presents an analysis to the protocols previously described in 2.2.2 in the light of the parameters defined in 2.2.1. It will help this research in two ways. Firstly, to have a general map that shows whether or not a specific protocol considers a parameter (*e.g.*, end-to-end delay), in such a case the middleware should have a way to interpret this information. Secondly, in case a protocol does not provide a parameter it is necessary for the middleware to create alternate functionalities in order to achieve such conditions when possible. The analysis is done by intersecting the studied protocols with the requirements. The evaluation is done as follows:

- Provided parameter (Yes): It means that the parameter is provided by the protocol.
- Non-provided parameter (No): It means that the parameter is not provided.
- Ambiguous parameter (NS): It means that it is not clear if the parameter is provided or not by the protocol.

Table 2-1 Analysis to data dissemination protocols

	End-to-End delay	***Energy efficiency***	***Transmission rate***	***Confirmation mechanisms***	***Congestion Control***
CCBs	Yes(*)	Yes(*)	Yes	No	No
DD	No	Yes	Yes	NS	Yes
TTDD	No	Yes	NS	No	Yes
GRAB	No	No	NS	NS	NS
FDDDP	No	Yes	Yes	NS	Yes
DDDP	No	Yes	NS	No	Yes
CBDDP	No	No	NS	NS	NS
EAGDDP	No	Yes	NS	NS	NS
(*) It means that the parameter changes according the protocol configuration.					

As can be seen in the previous table, no protocol fully considers the general parameters. Hence, the middleware should offer mechanisms for the completeness of the parameters when the selected protocols do not offer a certain feature. CCBs-protocol appears to be a suitable solution when the information should be disseminated with delay constraints. However, it is known that when end-to-end delay requirements are high the energy efficiency is low. DD and FDDDP are protocols where confirmation mechanisms parameter is not defined. Nonetheless, due to the packet reinforcement processes it seems to have this feature but it is not completely clear, then, it cannot be assumed. EAGDDP presents a similar characteristic in control congestion parameter. Therein, the literature describes no flooding due to the sending of information to a specific destination. Hence, it could be seen as an indirect congestion control mechanism.

2.3 Architectures

Similar to the previous section, the parameters of the architecture to be built are presented. Then, some middleware and framework solutions recently developed will be presented. Finally, the reviewed solutions are evaluated in the light of those parameters.

2.3.1 Architecture parameters

A middleware should not delegate all the data dissemination responsibilities to the protocols for either two reasons. 1) The protocol used by the middleware might not offer full data dissemination support as analyzed in the previous section, in such a case the middleware needs to take other decisions, in order to convey the information to the destinations. 2) When the protocol offers these capabilities, the middleware has to have a way to interpret and administer them (*e.g.*, end-to-end delay, confirmation mechanisms). That is why the middleware should have a way to be adapted to the protocols used in a specific environment, without being strictly dependent of them. In order to offer such independency the middleware should focus on the inclusion of several parameters. Bulasubramanian *et al.* [11] present four key parameters that a middleware for load balancing should have. This approach has been used and adjusted to define the architectural parameters of this research. Those are below detailed.

- *General purpose.* Although we are considering data dissemination as a principle, no assumption should be made regarding the applications that will use the

architecture and the protocols used to convey the information towards the destinations, *e.g.*, sensor nodes, gateways and base stations).

- *Transparency*. Applications using the middleware should not be aware of implementation details used inside; generic interfaces exposing the services should be provided.
- *Adaptive.* According to the application requirements, the middleware should adapt certain parameters to fulfill them, *e.g.*, maximum end-to-end delay and destinations. Rules and priorities should be considered in order to define the importance of messages when compared with others to be sent at the same time.
- *Extensible*. Middleware components and functionalities should be possible to be extended if required by applications. The middleware can incorporate new data dissemination protocols in order to be adapted to the changing network environments.

Unlike the parameters for data dissemination previously discussed, the parameters for the architecture should be always considered independently from the type of application.

2.3.2 Middlewares

Middleware design is a hot topic in the research community [7, 10, 11]. Numerous propositions have been presented to resolve common problems, such as QoS as reported by [11-13]. However, the majority of these solutions were thought to run on traditional platforms (*e.g.*, Internet, Cellular Networks). In contrast, propositions such as [7, 10] in turn designed to run over ad-hoc technologies (*e.g.*, WSN, VANETs) do not treat the problem of QoS, one of the main focus of this dissertation. Nevertheless, a review of this literature is done due to the importance and the relevance some features offer to this study. Accordingly, a description of the principal features of these works is given focusing on the aspects that may bring ideas to the design of this proposal, as it is the main goal of this dissertation.

The first review is made to the work of Balasubramanian *et al.* [11]. They present *Cygnus*, an open-source middleware developed to support adaptive and non-adaptive load balancing strategies. At run time, it can reconfigure these strategies in a transparent way for clients and servers. *Cygnus* is a Common Object Request Broker Architecture (CORBA) implementation

that presents four components. Firstly, a *load balancer* is responsible for optimal distribution of workloads across groups of servers. Then, a *member* component is a duplicate of a particular object. Next, there is an *object group*, which contains a set of objects that do the same remote operations. Finally, a *session* entity manages the time that a remote operation takes.

Moreover, *Chameleon* is another middleware proposition. It is a peer-to-peer middleware for self-organizing decentralized web services that was developed by Adam and Stadler [12]. It has three features that are common in peer-to-peer systems: 1) a server cluster has identical functional nodes, 2) the design is decentralized and 3) a partial view of the system is kept by each node. Furthermore, *Chameleon* supports QoS objectives and service differentiation. For such a purpose, it uses an epidemic protocol for data dissemination and for control information. It has a penalty model when a violation of QoS occurs, which exponentially increases each time. The environment, over which this middleware is executed, is a traditional Internet architecture. The authors use Tomcat as a web server, thereby, they install the TCP-W benchmark, which is a web application containing various servlets that execute common operations such as dynamic Hyper-Text Markup Language (HTML) and input/output (I/O) database operations.

In the same vein, Ribeiro *et al.* [7] present a message-oriented adaptive middleware, called *SensorBus*, that runs on WSNs. It is composed by two main services: a *message* service and *context* service. The *message* service provides communication and coordination for distributed components, whereas the *context* service manages different sensors that obtain data from several phenomena. Additionally, there is another service responsible for providing a single interface for application that is called *application* service. An important concept exposed by this work is the inclusion of editions. The middleware is developed having several editions. One particular edition is designed to be executed on a single type of device. There is an edition that runs on Personal Computer (PC) environments. Another edition was built to be executed on each sensor of the network. Finally, an interconnection module which is another edition; it is responsible for bridging the previous two. For routing purposes the middleware uses two well-known multi-hop routing protocols: Link Estimation and Parent Selection Method (LEPSM) and MiniRoute. The performance evaluation was done using the *surge* application which comes with TinyOS. The main goal of this process was to verify the impact of the routing protocols selection in the middleware. This evaluation leads the researchers to an important conclusion. Both

protocols present a similar performance, however this could be given by the chosen network topology.

Also, Martin-Escalona *et al.* [13] present MILCO, a middleware, especially conceived for cellular networks, reduces resources consumption and optimizes location traffic load. It also presents an appropriated latency. One of its major advantages is that it provides a required QoS by providing an optimum location technique for each request done inside the cellular network.

Finally, Delicato *et al.* [10] present a middleware family for VANETs. The middleware family is made up of three different viewpoints: *device*, *network* and *application* according their constraints. In order to manage each individual configuration, a Software Product Line (SPL) approach is used. It facilitates the development process whereas respecting quality restrictions. It emphasises in the application services due to the challenging requirements in vehicular networks. Even though this paper is not intensive in the definition of the network components, it presents in a decent manner the SPL methodology, when designing middlewares that run on heterogeneous environments, where quality needs to be assured.

2.3.3 Frameworks

A framework is a collection of classes, applications, libraries and Application Program Interfaces (APIs) to help different components to work together [5]. Besides the middleware propositions previously presented, frameworks have been also developed. They have similar goals to middlewares if comparing the requirements they want to reach. In this vein, a framework solution for data dissemination in WSNs is presented by Saha and Matsumoto [5]. It uses a protocol for data collection in disaster migration and rescue operations that presents low delay while improving energy consumption.

The data collection framework is divided into two subsystems: the disaster migration and the rescue operation. The first one is responsible for the tracking of a disaster (*e.g.*, an earthquake or a tsunami). This part is analyzed by the authors but not covered by the solution. The responsibility of the second one is to coordinate the rescue operation after a catastrophe occurs. Since the disaster can be originated underwater, as part of the architecture, researchers consider routing algorithms for delay-sensitive applications Under Water Sensor Networks (UWSN). In addition, the architecture also has terrestrial WSN support for disaster migration and rescue

operation. The first subsystem uses a data dissemination technique based on a hybrid model composed by sensors, Ad hoc Relay Stations (ARSs) and a cellular network. Sensors are responsible for collecting the information in a hybrid network, *i.e.*, it is composed by a cellular network and several ARSs. Relay stations are the back up of base stations when they are not available. ARSs have two interfaces. 1) To communicate with pairs or with sensor network 2) to communicate with base stations of cellular network. The deployment of the sensors is done in a preconfigured manner. The addressing scheme is done using polar coordinates with respect to the base station. The data is disseminated from each sensor to a *cnode*. Herein, the *cnode* broadcasts a task message including its own coordinates. When the nodes receive the message from the *cnode*, they proceed to calculate their coordinates. By having this information, each node can send information to the *cnode*. For this purpose, it may be necessary to use a multi-hop communication; therefore, the sensor node selects the next hop using minimum angular deviation relative to the *cnode*. This framework uses implicit positive acknowledgement (ACK). Once a node sends a message, it waits for repletion of the packet. In case of failure, it broadcasts the message using its maximum transmission power.

The performance evaluation of this protocol was done by comparing it with Sensor Networks for Disaster Relief Operations Management (SENDROM). Results show that energy is better used by this approach. Additionally, the failure ratio is lower. Furthermore, the delivery rate is also improved. Finally, even though the researchers argue that the delay is outperformed they do not have any particular value to verify this assertion.

2.3.4 Analysis to middlewares and frameworks

Similar to the analysis made to the protocols, the middlewares and frameworks are evaluated. The evaluation is done in a range scale as follows: High, Medium, Low, Nothing. Each of these values expresses how well the requirement is met by the protocol. The following table shows the result of such analysis.

Table 2-2 Analysis to middlewares and frameworks

	General purpose	***Transparency***	***Adaptive***	***Extensible***
Cygnus	Low	High	Medium	Medium
Chameleon	Medium	High	High	ND
VANET-Middleware	High(*)	High	Low	ND
MILCO	Low	ND	High	High
SensorBus	High(*)	High	Medium	ND
Disaster Management Framework	Medium	Nothing	High	Low
(*) the architecture considers editions.				

SensorBus and *VANET-Middleware* appear to be the architectures that offer better support for the parameters previously exposed. However, both of them miss important details. On one hand, *VANET-Middleware* is especially conceived to run on top of Vehicular Networks. This is the reason why its extensibility and adaptability are its major weaknesses. Additionally, the definition of the services is not given in detail. Therefore, it seems to be quite vague in some parts, particularly considering the network view. On the other hand, *SensorBus* runs on WSNs. It presents a better adaptation than *VANET-Middleware*. Nevertheless, their authors do not give any detail concerning its extensibility. They work specifically with two fixed protocols (LEPSM and MiniRoute). Despite of their limitations, these approaches offer several important architectural concepts that can be taken into consideration for the foremost decisions of this research. However, from the data dissemination point of view and their interactions with the protocols there are, indeed, several aspects that should be better addressed. The next work then, is to

include such improvements in the proposition in order to have an integrated architecture that incorporates the parameters from data dissemination and architecture points of view.

2.4 **Parameters integration**

After having analyzed the parameters for data dissemination protocols and those for the architectures (*i.e.*, middlewares and frameworks) in the previous sections, their integration is necessary since they need to be considered during the middleware design process. Accordingly, Figure 2-2 summarizes the parameters from the two perspectives: data dissemination and architecture.

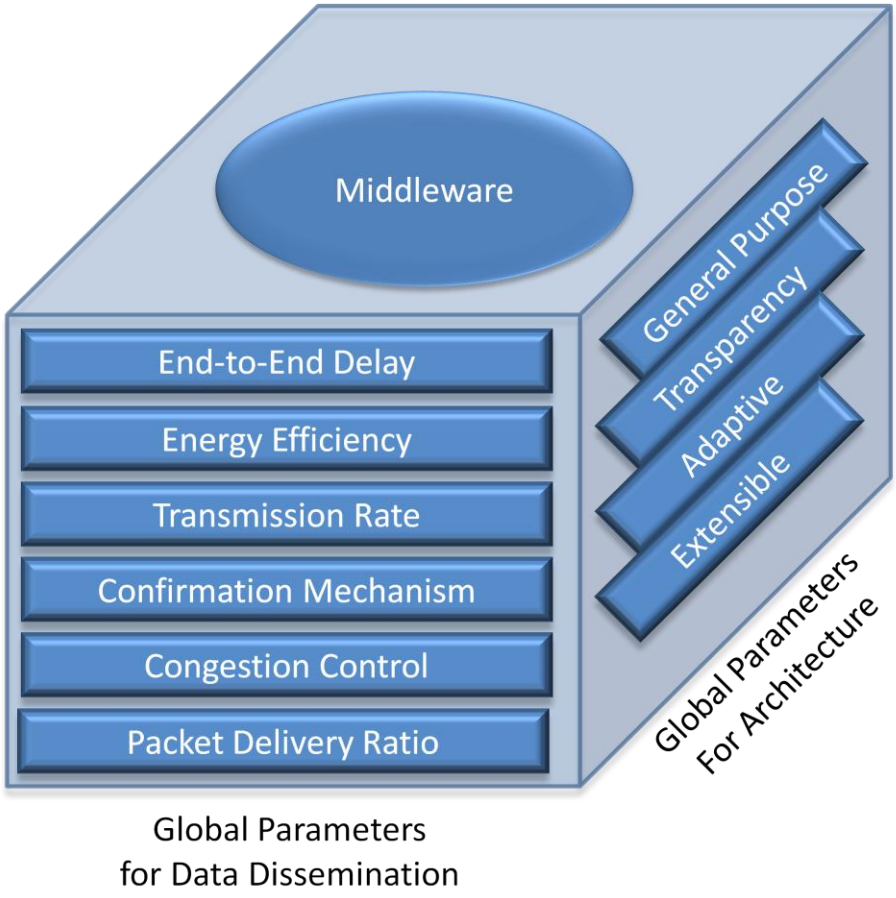

Figure 2-2 Parameters Integration

As it can be seen, there are three dimensions. The Data Dissemination dimension shows all the parameters that the middleware needs to consider from this perspective. Similarly, the

Architecture dimension presents the parameters that should be supported. Finally, the middleware dimension represents the ideal solution to be built which results from the intersection between the parameters for the data dissemination and architecture point of view. A final analysis that can be done is that each parameter from the data dissemination dimension is related to each parameter from the architecture dimension. It means that each data dissemination parameter (*i.e.*, end-to-end delay, energy efficiency, transmission rate, confirmation mechanism, congestion control and percentage of success) should be for general purpose, transparent adaptive and extensible.

The next chapter, thus, focuses on presenting a detailed architecture that will assure the achievement of the research goals. Particularly, it mainly focuses on the inclusion of all the architectural and data dissemination parameters analyzed in this chapter.

Chapter 3 PROPOSED ARCHITECTURE

Any architecture needs to express in an appropriated and a clear way the main layers and components of a system [14], including its software elements, relations among them and their properties. In order to facilitate the definition, the understanding and the validation of the system, a paradigm is usually required. It assures a unique interpretation of the system where ambiguities are avoided. Since this thesis proposes a middleware-architecture for data dissemination in WSNs with QoS requirements, its analysis, design and test should be done using a model that facilitates these intensive processes. Therefore, a deep understanding of the selected software development paradigm is imperative.

The rest of this chapter is organized as follows. The first section presents the definition of the software development paradigm to detail the architecture. The subsequent sections focus on the middleware-architecture specification using such paradigm from two different engineering perspectives.

3.1 Software development paradigm outline

As stated before, a middleware is a software component that connects other components or applications. The middleware, presented in this chapter, deals with different data dissemination protocols and it requires to be executed on different types of devices (*i.e.*, sensor, gateway, base station), which forces each environment to control different configurations and specificities. To facilitate the design process of such middleware, Software Product Line Engineering (SPLE) has been considered [14], which is a paradigm that supports the software engineering process. It controls the different configurations that a middleware may have, when running on different platforms, devices and networks while preserving commonalities, *i.e.*, common features to all components involved [10]. SPLE is made up of two phases in order to guarantee an accurate configuration of the system while ensuring its quality: *Domain Engineering* (DE) and *Application Engineering* (AE), as depicted in Figure 3-1 (taken from [14]).

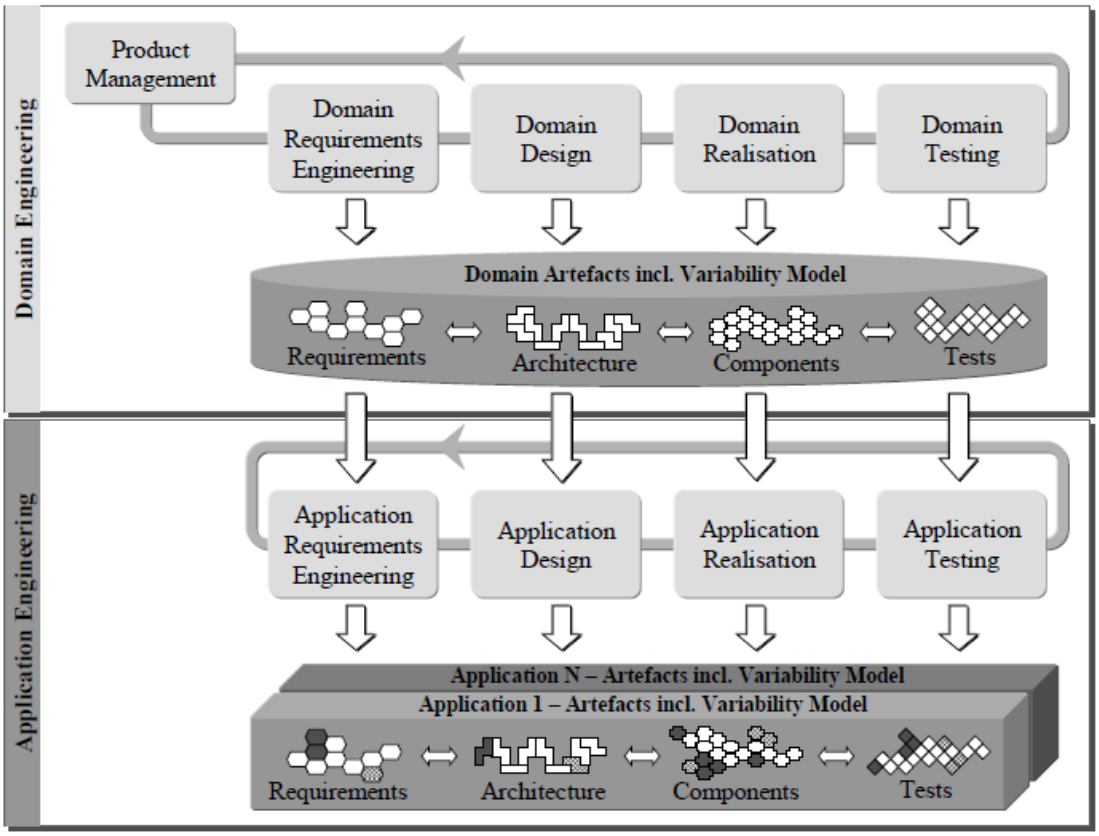

Figure 3-1 Software Product Line Engineering Framework

Firstly, DE aims to define the commonality and the variability of the system. It is useful when different configurations are required. It uses several artifacts (*i.e.*, requirements, architecture, components and tests) to represent the domain. These artifacts are included in DE sub-processes. *Product Management*, which is the starting point of the whole process, deals with economic and market aspects, and produces the roadmap product, which is used for other sub-processes later. Additionally, the *Domain Requirements Engineering* uses this roadmap for the elicitation and documentation of the requirements. Then, the reference architecture is defined in *Domain Design*, using the requirements gathered in the previous sub-process. Afterwards, *Domain Realisation* provides the detailed design and implementation, focusing on reusability as a key concept. Finally, the validation and verification is done in the *Domain Testing* stage.

Subsequently, when DE is complete, the AE process is started. It is intended to produce the Product Line Architecture (PLA), *i.e.*, it contains all required components to implement a product of Software Product Line (SPL) family and gives the middleware a specific configuration based on the specific requirements and constraints of each environment. For that reason, the

middleware reuses DE definitions while exploiting the product line variability. AE sub-processes are the same as in DE. They use the artifacts produced in DE sub-processes as entry points.

The following sections are focused on presenting the middleware proposal following the SPLE framework. Initially, the *Domain Engineering* phase of the middleware is depicted in sections from 3.2 to 3.6. Since the main goal of this chapter is to present de architecture, *Domain Testing* is not included. Once *Domain Engineering* is concluded, the focus goes to the *Application Engineering* phase beginning in section 3.7 until the end of the chapter. It specifies the particularities of each product line that is intended to be developed.

3.2 **Product Management**

As it was previously discussed, the main goal of the *Product Management* sub-process is to produce the roadmap of the middleware. Accordingly, Figure 3-2 presents a general overview of such system; which helps to the roadmap definition. Therein, the middleware is initially related to *delay-constrained applications*, as this work aims to produce support for such type of applications. In order to guarantee this support, it requires a direct communication with *data dissemination protocols* at any moment. Therefore, the top layer represents *delay-constrained applications* that use the *middleware* which is the intermediate layer. In the meantime, it imposes some requirements (*e.g.*, end-to-end delay) to the underlying *data dissemination protocols* located in the bottom layer.

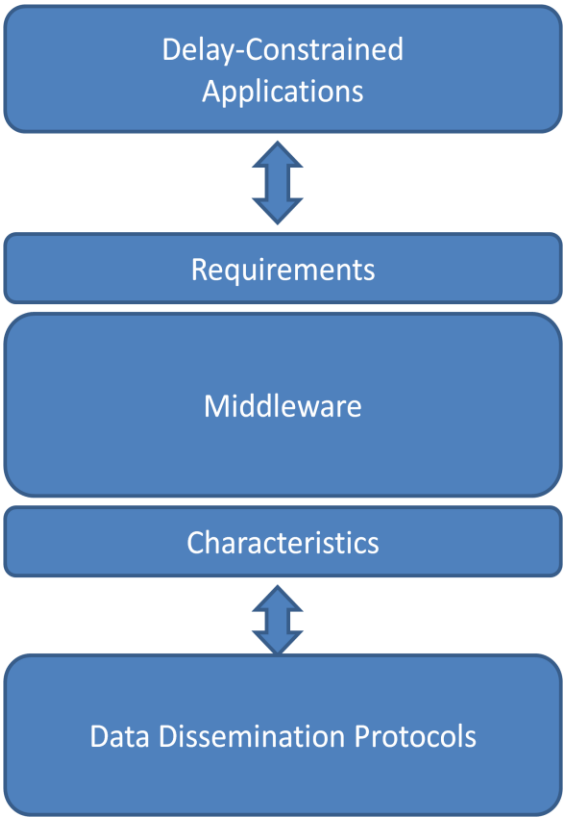

Figure 3-2 Middleware General Overview

Having defined the relation between the applications, the middleware and the data dissemination protocols, the next step is to present the roadmap definition. For such a purpose, Delicato *et al*. [10] approach is adopted. It considers three points of view as defined in Chapter 2: *device*, *network* and *application*. Firstly, *device perspective* focuses on each device and its inner components. This perspective does not consider any interaction with the environment (*e.g.*, other devices). Herein, the middleware deals with important features, such as type of devices and operating systems. Secondly, *network perspective* represents the dissemination of information among the network. Therefore, it takes into account several network characteristics, *e.g.*, end-to-end delay, confirmation mechanisms and energy optimization. Finally, *application perspective* represents the applications using middleware services. As defined above, these applications impose delay constraints to the middleware.

The integration of the previous three perspectives constitutes the roadmap of this proposition, guarantying a holistic view of the system. This amalgamation is intended to show

that all perspectives are present at any time in the system and the intersection of all of them produces the middleware. Figure 3-3 depicts such integration.

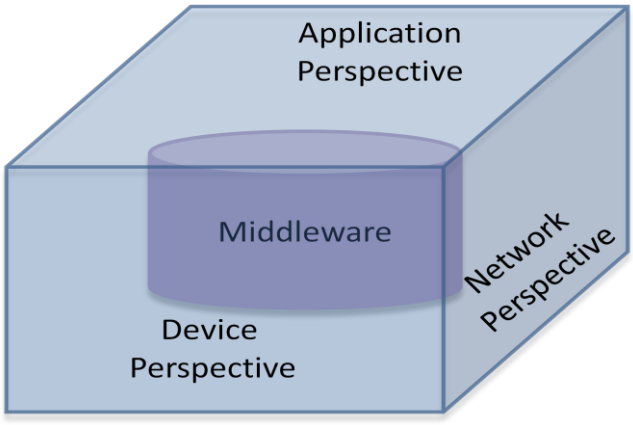

Figure 3-3 Middleware Roadmap

The 3D-view offers the possibility to analyze the system from different perspectives, while preserving the unity and respecting requirements and constraints. The challenge is to provide an architecture that keeps correct configurations in every single environment where the middleware is deployed. Each configuration, as it will be further explained, is known as an *edition*.

3.3 Domain Requirements Engineering

As the *Product Management* phase has been defined, the second step is the definition of the domain requirements. This section describes each requirement, considering the roadmap previously presented, from functional and non-functional perspectives. Functional requirements describe all functionalities the system is supposed to accomplish, while non-functional requirements focus on operational requirements instead of behavioral requirements and serves as a support for the first category. Functional requirements are defined in the light of the parameters for data dissemination defined in section 2.2.1. Likewise, non-functional requirements are presented according to the parameters discussed in section 2.3.1.

3.3.1 Functional Requirements (Data Dissemination)

End-to-End delay

REQ 1. The dissemination process shall be delay-constrained. It means that the total time a message takes to reach a target shall not exceed a time-constraint (*e.g.*, a node X must send a message to a User Group member in $t \leq 5$ sec).

REQ 2. The dissemination process shall consider different data dissemination protocols. When the maximum time set for a message expires (*e.g.*, $t \leq 5$ sec), the middleware shall select an alternative protocol.

Adjustable transmission rate

REQ 3. The system shall offer an adequate transmission rate. It means that the system shall manage priority levels (*e.g.*, 0=emergency, 1=informational) which can affect the rate. The messages with higher priority should be delivered first.

Confirmation mechanisms

REQ 4. Record a detailed log of sent, received and read messages. The system shall keep a record of the messages sent through different protocols. Furthermore, it shall know whether those messages have been successfully received and acknowledged by the destinations.

REQ 5. Each single message sent shall be tracked at any moment. Accordingly, the architecture shall update the message with a positive acknowledgement (*i.e.*, ACK) when the message is successfully received by the destination. In such a case, the system shall not do anything else. In contrast, a negative acknowledgment (*i.e.*, NACK) is produced upon unsuccessfully reception of a message, thus, a reactive strategy is activated (*i.e.*, select another data-dissemination protocol). If no response is obtained from the destination in a maximum period of time (*e.g.*, $t \leq 5$ sec), then a no-response status (*i.e.*, NR) is set in the system to that message, in such case the system proceeds by selecting another data-dissemination protocol.

Percentage of Success

REQ 6. There should be a control between the message sent to the destination and its status, *i.e.*, ACK, NACK and NR. The system should have an optimality superior to 95%; which means that from 100 sent messages at least 95 should be successfully received. Non-Functional Requirements (Middleware)

General purpose

REQ 1. The system shall be designed to be independent from the data dissemination technique.

REQ 2. The system shall be designed to manage different environmental values. The meaning of these values is defined by users through set up parameters.

REQ 3. The system shall be designed to run on several platforms, such as sensor nodes, gateways and base stations.

Transparency

REQ 4. The middleware shall be independent from the application that will use it. Therefore, functionalities shall be exposed by using standard access techniques (*i.e.*, interfaces).

REQ 5. The middleware shall be designed hiding implementation details to the applications. However, it should provide feedback about what is happening on the network, avoiding unnecessary details.

Adaptive

REQ 6. The system should be able to adjust the values used in a dissemination process depending on priority levels. The main adjustable parameter is the maximum message delay for each specific protocol.

Extensible

REQ 7. The system should be designed to easily incorporate new functionalities (*e.g.,* new data dissemination protocols could be added)

3.4 **Mathematical approach for end-to-end delay evaluation**

Since the end-to-end delay is one of the most important domain requirements that has to be fulfilled, as it was presented in the previous sub-section, this part of the work focuses on defining a mathematical approach, in order to understand its behavior within a data dissemination process. Accordingly, in networking, every communication has at least two actors: the *originator* and the *terminator* connected by a network. The first one sends a message and the second one

receives it. This process is a simple interaction between two nodes. However, sometimes this simple process could raise a more complex communication path, when forwarders or intermediate nodes are required, *i.e.*, data dissemination from A to F as shown in Figure 1-2. It can be seen that in each segment, both actors, *i.e.*, the *originator* and the *terminator*, can be identified. For instance, in $delay_1$ the *originator* is A, whereas the *terminator* is B. The latter may send back an ACK upon successfully completion of a message reception, or a NACK otherwise. In this example, $delay_1$ represents the time elapsed between A and B when communicating and exchanging messages. When a negative acknowledgement or no response is obtained, this delay is higher.

In order to quantify the end-to-end delay for the whole communication, each segment needs to be analyzed (*i.e.*,$delay_1$, $delay_2$, $delay_3$, $delay_4$ and $delay_5$). For this purpose, Figure 3-4 presents the five delay-segments required for each *originator* and *terminator* to exchange a message as follows:

T_1- The middleware execution time in the *originator* side.

T_2- The delay in the communication between the *originator* and the *terminator*.

T_3- The middleware execution time in the *terminator* side.

T_4- The time elapsed from the answer to arrive from the *terminator* to the *originator*.

T_5- The service lookup delay when NACK or no response is obtained.

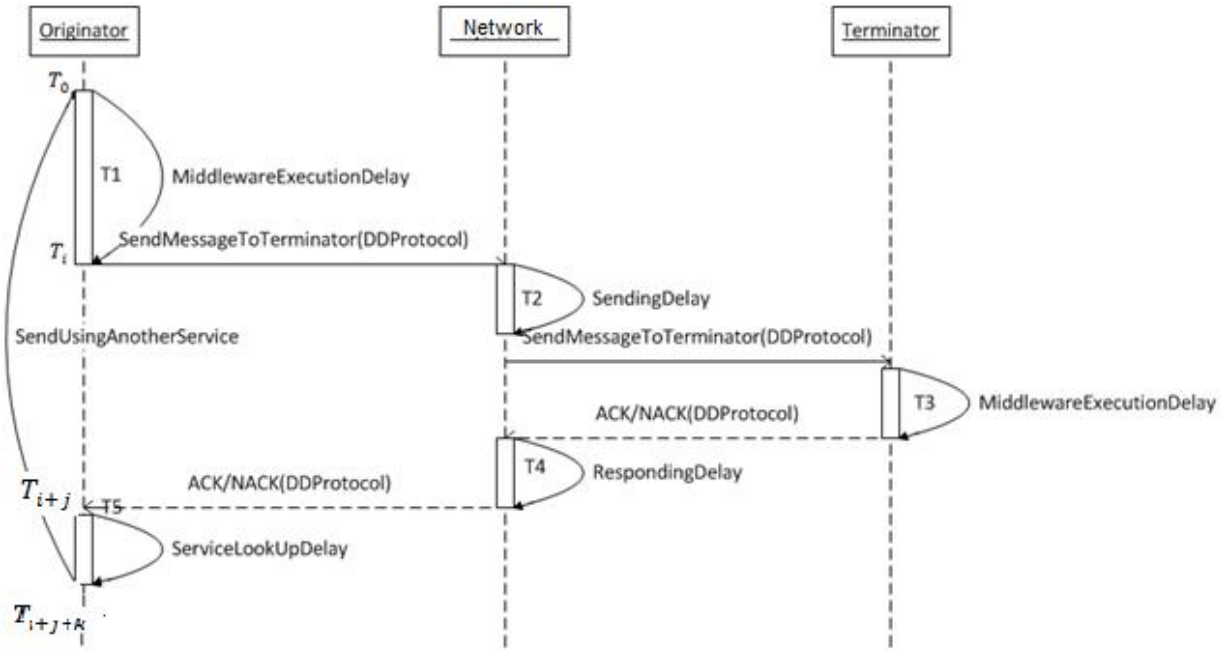

Figure 3-4 Delay analysis for each originator and terminator

In this sequence diagram three possible scenarios might happen. 1) The *originator* could receive a positive response from the *terminator* in a valid period of time, *i.e.,* ACK, 2) the *originator* could receive a negative response in a valid period of time, *i.e.*, NACK or finally, 3) the *originator* could not receive any response from the *terminator, i.e.,* NR. In each scenario, the delay is completely different. Each one of these scenarios and the total delay will now be further analyzed.

3.4.1 Delay between originator and terminator with positive response (ACK)

This scenario considers the case when the *originator* receives an on-time and positive acknowledgment from the *terminator*, concluding the communication process between both entities. Thereby, the delay can be represented as the total time elapsed in the middleware execution (*i.e.,*T_1) and the delay to receive a response from the *terminator* (*i.e.,*T_2, T_3, T_4).

$$delay_{ACK} = delay_{middleware} + delay_{response} \quad \textbf{(3.1)}$$

As can be seen in Figure 3-4, the middleware instance executed on the *originator* side takes a time T_1 to complete the message sending process. This is considered as the time elapsed

from the beginning of the middleware execution, up to the point where the middleware reaches the dissemination protocol that will send the message through the network. Let us assume that the middleware begins its execution at T_0 and it requires *i* time units to reach the dissemination protocol component. The delay is then the difference between these two times and can be expressed as follows:

$$delay_{middleware} = T_i - T_0 \quad \textbf{(3.2)}$$

Once the dissemination protocol is called, the response delay begins. This delay can be defined as the difference between the time when the middleware receives a response from the dissemination protocol and the time when it was originally called. For instance, let us say that the middleware calls the dissemination protocol at T_i and *j* time units later it receives an answer, thus, the dissemination protocol response is said to be received at T_{i+j}. Then, the response delay in this case is the difference between these two times and it can be expressed as follows:

$$delay_{response} = T_{i+j} - T_i \quad \textbf{(3.3)}$$

Now, taking the delay ACK formula (3.1) and replacing it by middleware (3.2) and response (3.3) delays, the following result is obtained:

$$delay_{ACK} = (T_i - T_0) + (T_{i+j} - T_i) \quad \textbf{(3.4)}$$

$$delay_{ACK} = T_{i+j} - T_0$$

3.4.2 Delay between originator and terminator with negative response (NACK)

When the *originator* receives an on-time negative acknowledgment, it means that the message was received by the *terminator;* but it was an unsuccessful delivery. It might arise when having corrupted or incomplete messages. In this case, the message needs to be sent again by another protocol. The delay is quite similar to the one previously analyzed. Nonetheless, an extra variable should be considered. The delay can be represented as follows:

$$delay_{NACK} = delay_{middleware} + delay_{response} + delay_{lookup} \quad \textbf{(3.5)}$$

The lookup process is supposed to be executed immediately after the middleware receives a negative acknowledge from the *terminator* and takes *k* units to be finished. Its representation is shown below:

$$delay_{lookup} = T_{i+j+k} - T_{i+j} \quad \textbf{(3.6)}$$

Similar to the previous section, taking the delay formula (3.5) and replacing it by the all delays (3.2, 3.3 and 3.6), the following result is obtained:

$$delay_{NACK} = (T_i - T_0) + (T_{i+j} - T_i) + (T_{i+j+k} - T_{i+j}) \quad \textbf{(3.7)}$$

$$delay_{NACK} = T_{i+j+k} - T_0$$

As can be seen, there is an additional time (*i.e., k* time unites) that is required. However, this time does not consider the additional delay required for the middleware and the network to send the message through a data dissemination protocol.

3.4.3 Delay between originator and terminator with no response (NR)

This scenario is required when no response is obtained from the *terminator* or the network. In such case the *j* time units required by the dissemination protocol to finish the task becomes infinite. Therefore, it is imperative to count with a component that breaks this state. The system is then forced to look for another way to send the information when *j* exceeds a threshold defined in each middleware instance. The delay for this scenario considers three elements. Firstly, similar to the previous two scenarios, it has a middleware delay. Secondly, the maximum network delay is required, since no response is received from either the network or the *terminator*. Finally, the lookup delay, which is basically the same described in section 3.4.2 to find another service to send the information towards the target.

$$delay_{noresponse} = delay_{middleware} + delay_{max} + delay_{lookup} \quad \textbf{(3.8)}$$

Accordingly, the delay maximal is

$$delay_{max} = \alpha_j \quad \textbf{(3.9)}$$

where α_j is a maximum time for a dissemination protocol to fulfill a j-task.

The delay can be finally presented as follows:

$$delay_{noresponse} = (T_i - T_0) + \alpha_j + (T_{i+\alpha_j+k} - T_{i+\alpha_j}) \quad \textbf{(3.10)}$$

Under this particular scenario, the total delay, which will be further discussed, will be even higher. Since the middleware needs to look for another data dissemination protocol to send the information, some additional time is required.

3.4.4 Total delay between the originator and the terminator

As the three possible scenarios have been analyzed for each segment (S), it is now possible to calculate its total delay from the *originator* to the *terminator*. This delay can be represented as the sum of one delay with positive response (ACK), the total of all the delays with negative response (NACK) required and the total of all the delays with no response (NR). The following equation shows this delay:

$$delay_S = (delay_{ACK}) + \sum_{i=1}^{w}(delay_{NACK})_i + \sum_{j=1}^{x}\left(delay_{noresponse}\right)_j \quad \textbf{(3.11)}$$

where *w* is the number of transmission with negative response and *x* is the number of transmissions without response.

It is important to point out that the $delay_{ACK}$ is considered only once, meaning that if an ACK is received, no retransmissions using other protocols are required. In contrast, if a NACK or NR status is reached, the information is disseminated using other protocols, *i.e.*, using the lookup process, which increases the total delay.

3.4.5 End-to-end delay

In order to calculate the end-to-end delay, *i.e.*, time elapsed to send the information from the very first *originator* to the last *terminator* each segment delay needs to be considered. Therefore, the following equation shows this delay:

$$delay_{end-to-end} = \sum_{S=1}^{N-1} delay_S \quad \textbf{(3.12)}$$

where N is the number of nodes (forwarders) that the information uses along the process.

This equation considers all the delays previously analyzed in each segment. Furthermore, it considers all possible scenarios that might happen in each pair of nodes as it was explained in the previous sub-sections.

3.5 Domain Design

Up to this point, the architecture roadmap, functional and non-functional requirements and the end-to-end delay along the communication path have been explained in detail. In this subsection, all these elements are considered as the starting point to present and describe the design of the architecture. In the light of this information, the reference architecture is presented.

3.5.1 Feature model

The feature model is presented using the roadmap previously defined. Therein, the three perspectives of the middleware are further analyzed. Initially, the *device perspective* focuses on each device and its components, considering five features: *type of devices*, *operating systems*, *radio technology*, *development technologies* and *storage*. *Type of devices* represents different machines where the middleware is intended to be executed (*i.e.,* sensor node, gateway, base station). *Operating systems* represent different operational platforms running on the types of devices (*i.e.,* TinyOS, Linux and Windows). *Radio technology* is used to establish communication with other nodes of the architecture (*i.e.,* 802.11). Additionally, *development technologies* features need to be taken into consideration. For the suitability of these technologies and their widely acceptance in the academia, nesC, Java and C++ have been chosen. Finally, *storage* takes care of the persistence of the information when needed (*i.e.,* databases, XML files).

Then, the *network perspective* takes into account three network services in order to achieve the requirements: *delivery manager*, *message sender manager* and *service manager*. Firstly, *Delivery Manager* (DM) is responsible for managing the delivery process. It tracks sent messages along the process, while considering already imposed time constraints. It includes: reporting, receiving and analyzing capabilities. Secondly, *Message Sender Manager* (MSM) is in charge of the message sending process. It is made up of three main processes: listening, analyzing and sending. Finally, *Service Manager* (SM) is a service that allows managing the protocols the system works with and the resources associated to each of them.

Finally, the *Application perspective* is divided into two main categories: *delay-constrained applications* (*i.e.,* the main focus of this proposal) and *user applications* (*e.g.,* marketing and intelligent transportation). On one hand, *delay-constrained applications* have strict QoS constraints and are used to warn people in emergency events. They require a continuous

feedback from the middleware. On the other hand, *user applications* tolerate lower QoS constraints due to their specific goals. Failures or delays are not as critical as they are for the former category. Therefore, priorities are usually lower. Confirmation mechanisms could also be avoided or delayed. Despite of that, user applications can use the middleware as well.

3.5.2 Reference architecture

In this sub-section, the basic aspects that lead to the design of the reference architecture are presented. This reference architecture is intended to gather the different aspects that may vary from one product to another, but which have to be present nonetheless. Firstly, the architecture principles are depicted. They describe the roles in the system and an overview of the general architecture from the infrastructure point of view. Later on, the reference architecture is detailed.

3.5.2.1 Architecture principles

Roles: The architecture considers two roles. On one hand, there is the *Message Originator* (MO), which is responsible for initializing the notification process. On the other hand, there is the *Message Terminator* (MT), which receives the information and sends back a response. MT is a role played by any person or device in the system. In case of a person, it can be either a *Security Group* (SG) member or a *User Group* (UG).

Infrastructure Architecture Overview: The general architecture, as depicted in Figure 3-5, involves several infrastructure components. It integrates the WSN, as it was discussed in Figure 1-1, with two other networks. The source of information is the sensor network, whereas the destination can be Internet or a Cellular Network.

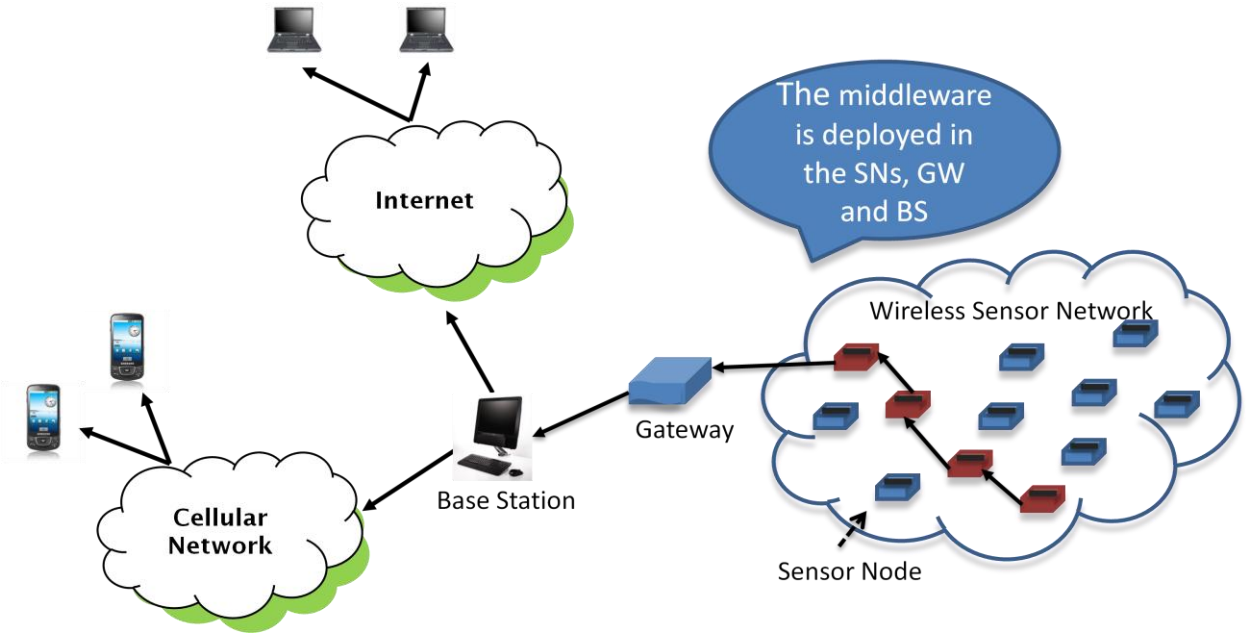

Figure 3-5 Global Architecture

The MO represents each single sensor node that is deployed. It collects information that could at some point be disseminated. If an event is detected, the sensor node starts the dissemination process towards the gateway, using the forwarders in between. The gateway is responsible for receiving information sent by any node in WSN and conveys it to the base station. Once the base station receives the information, it will make a decision depending on its own configuration, *e.g.*, Send information to UG and SG through different protocols, such as Short Message Service (SMS) and e-mail.

3.5.2.2 Middleware architecture

The architecture to be defined has to rely on a middleware that is installed in all main components along the system, *i.e.*, sensor nodes, gateway and base station, as shown in Figure 3-5, improving QoS and confirmation in the delivery process. The middleware, as depicted in Figure 3-6, is a three-layer architecture that has additionally three transversal planes acting upon all layers. The planes contain all the components that are general for all components of the middleware. In this part, *QoS*, *Confirmation* and *Timeliness* can be found. The layers are divided into three main categories: *Interfaces (IX)*, *Business Rules* (BR) and *Data Services* (DS).

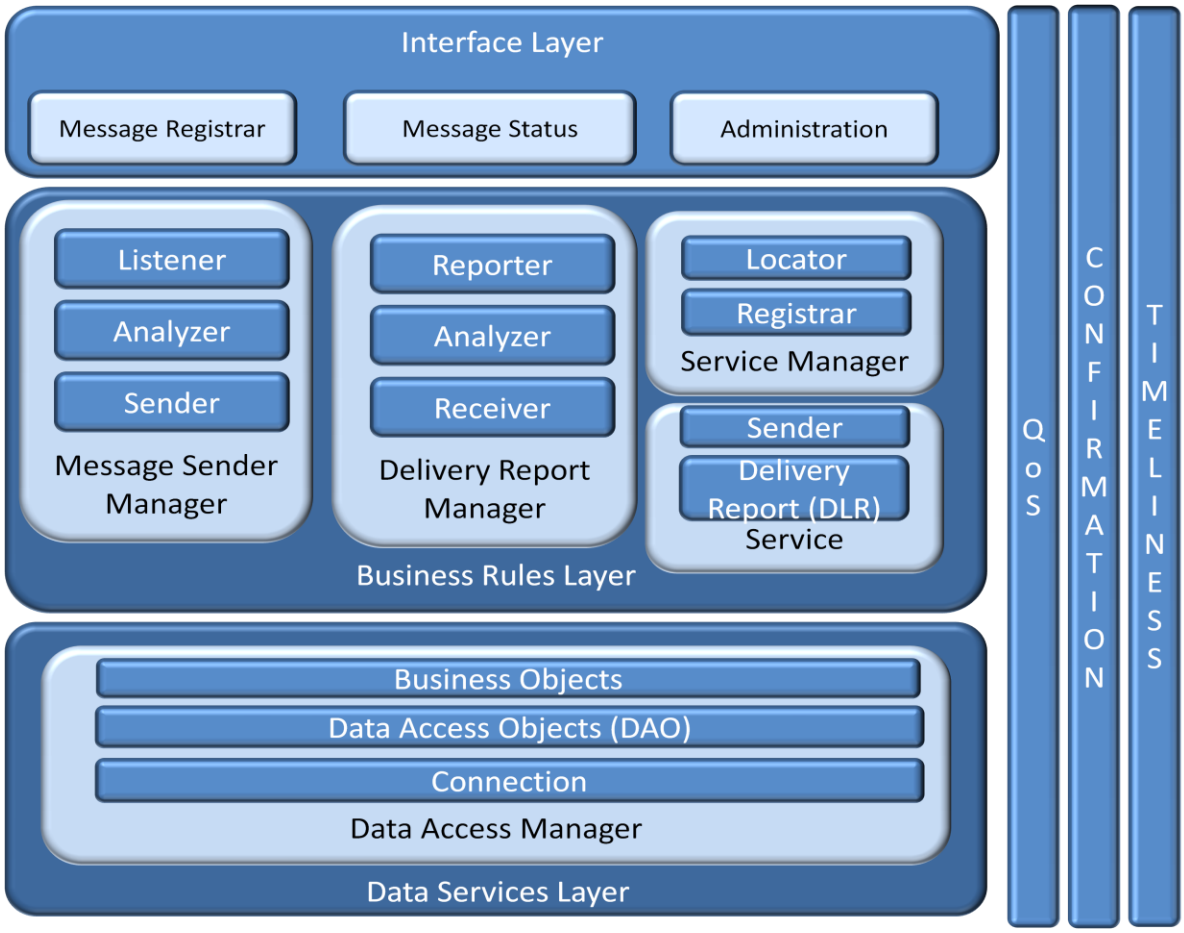

Figure 3-6 Reference Architecture

Concerning the *Interface* layer, there is a set of standard-access services that are responsible for exposing a unique way for any MO to register alerts (*i.e.*, *Message Registrar*), to collect information of each status (*i.e.*, *Message Status*) and to manage the data model (*i.e.*, *Administration*). Further defining these three services; firstly, the *Message Registrar* enables the possibility for MOs to register new messages to be sent according to the defined parameters (*e.g.*, priority, delay-constraints). Secondly, the *Message Status* collects history information related to the messages sent through the middleware in order to know their status. Finally, *Administration* allows the interaction with the *Data Services* layer in order to manage the data model, *e.g.*, users' information, groups' information, enabled protocols, priorities, delay-constraints, among others.

The second layer of the architecture is the *Business Rules* layer. It is the heart of the middleware and is composed by three processes: *Message Sender Manager* (MSM), the *Delivery Report Manager* (DM) and the *Service Manager* (SM). Firstly, The MSM is in charge of the message sending process. It has three main processes: *Listener*, *Analyzer* and *Sender*. *Listener* is

a process running in the background (*i.e.*, daemon process), which senses the messages coming to be disseminated in order to assign them to *Analyzer*. Thus, it queues the messages in a waiting list. *Analyzer*, which is another daemon process, multiplexes the messages by considering their priorities, according to the system configuration, *i.e.*, some users are notified before others. At this point, the system uses the information previously set up (*e.g.*, registered users, specific devices models, protocols, priorities, and message time intervals, among others). And *Sender* uses the result from the previous process to actually send messages to each MT. It relies on the SM and on the DM to perform the dissemination process. The second process of the Business Rules layer is the DM. It is responsible for managing the delivery process. Once a message has been sent to a MT, the DM tracks it along the process; considering the already imposed time constraints. For such a purpose, DM comprises three services: *reporter*, *receiver* and *analyzer*. *Reporter* knows each message status. It collects the history information from the messages and enables it for the interface module to access it. *Analyzer* is the responsible for doing any background task requested by *Reporter*. It considers the priorities, time delays and status reported by *Receiver*. This service interacts directly with SM to read each Delivery Report (DLR). Finally, the SM is a process that allows managing the different data dissemination protocols. It has been defined as a component that dynamically loads each resource (*i.e.*, an instance of a data dissemination protocol that manages its own parameters). Furthermore, more than one resource can be associated to each protocol, which gives the system the possibility to serialize a massive alert into several resources in the network (*e.g.*, using cluster servers in large-scale systems). It increases the chance for users to receive an on-time notification. Each notification is tracked using a unique standard called *Delivery Report* (DLR), which allows knowing if the MT receives the message.

The last layer, *Data Services* layer, manages the persistency of the middleware. It is responsible for administering the information of the data dissemination protocols and information regarding the messages. This information is stored mainly in two repositories: databases and XML files. The first one stores the massive information, such as user and group information. The second one contains several parameters to interact with the data dissemination protocol, such as component name, port number and maximum delay.

Having finished the three layers, an explanation of the planes is presented. The planes are modeled as logic views; *QoS*, *confirmation* and *timeliness* are treated independently in each

component but considered in some or all of them. They are oriented to offer a reference architecture that manages the end-to-end delay as a QoS parameter. Moreover, confirmation features are offered that permits to know the status of a message (*i.e.*, ACK, NACK or NR). Finally, timeliness support is done through the presence of daemon processes, (*i.e., Listener, Receiver and DLR*) that are executed at any time.

3.6 **Domain Realisation**

This section focuses on presenting the software components that lead to the deliverables of the domain realisation. Firstly, the class diagram that shows the classes interactions within the whole system including its interfaces is presented. Later on, the sequence diagrams that show the interaction of the architecture components are explained.

3.6.1 Class diagram

Figure 3-7 presents the diagram class of the system. It was divided in four logical layers, which depict the main components presented in the architecture reference. It presents a static view of the system. The first three layers refer to the five main components (*i.e., Interfaces, Message Sender Manager, Delivery Reporter Manager, Data Access Manager and Service Manager*) and the bottom layer represents the data dissemination protocols to be used. On top of the diagram, there is the set of *Interfaces* classes which offers a unique way for consumers to use the middleware services. It is made up of three classes that interact with the second layer components. Focusing on the second layer, the diagram can be read from left to right. Therein, it can be found *Message Sender Manager*, which, as explained before, is responsible for administering the sending process. It considers three main classes: *Listener*, *Analyzer* and *Sender*. The *Listener* senses new messages that arrive to the middleware. The second one is composed by four classes, which means that all classes need to participate in the process when the analyzer is executing. Finally, the *Sender* is in charge of sending the analyzed message. Then, the *Delivery Report Manager* classes track the message status. Similarly to the previous component, it also considers three classes: *Reporter*, *Analyzer* and *Receiver*.

Finally, in this layer, there is the *Data Access Manager*. It is responsible for providing and modifying the data models (*i.e.,* Databases and Configuration Files). It uses an *ActionController*, which is responsible for receiving an action to be executed and identifying which component in

the system will realize it. Normally, this action is assigned to a Data Access Object (DAO), which in turn, affects the information relying on any Business Objects (BOs), (*i.e.*, Priorities, Protocols, Users) required to succeed the task. The third layer of the architecture shows the *Service Manager*, which is responsible for interacting with the protocols and the network to complete either the message sending process or the delivery report process. It is composed by a set of classes that offer system characteristics, such as end-to-end delay (*ETEDM*), delivery report (*DLR*), environment events (*EnvironmentRecorder*), Confirmation features (*ConfirmationAgent*) and sending of messages (*IServices* and *IRessources*). *Service Manager* relies on a *ServiceLocator*, which identifies the most appropriate services and protocols according to the application requirements.

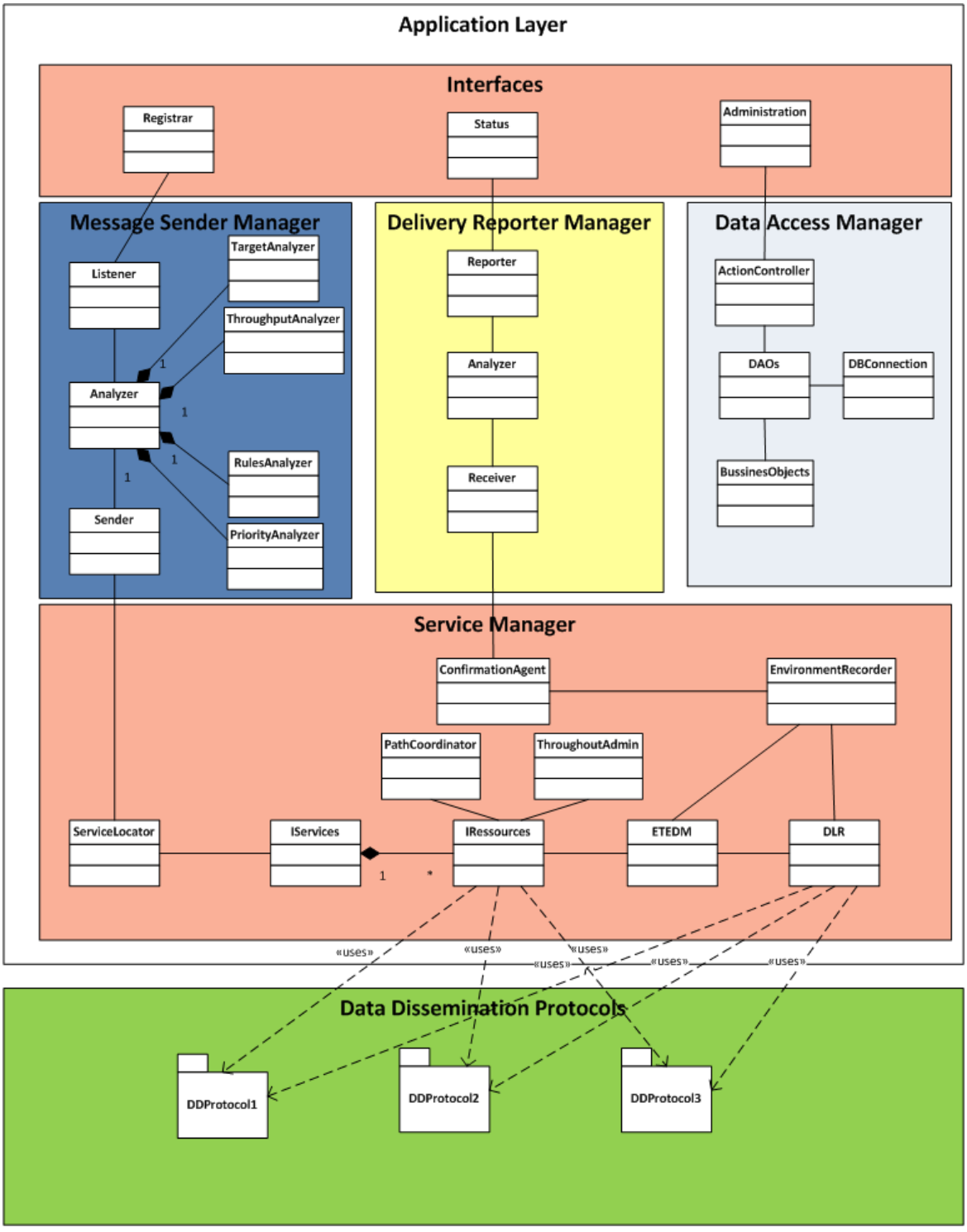

Figure 3-7 Class Diagram of Reference Architecture

As previously discussed, the middleware is located in the application layer. In order to perform its tasks, it should have access to specialized protocols that are normally located in lower layers in the communication stack (*e.g.*, data dissemination protocols). For such reason, the

bottom layer shows the available protocols to be used and their interactions with the middleware. It is important to notice that this proposal is protocol independent. It means that any protocol could be used as long as it supports the application requirements. Therefore, it is up to the implementers to choose the right dissemination protocol.

3.6.2 Sequence diagrams

As shown in the previous section, the class diagram focuses on a static view of the system; sequence diagrams, in contrast, present a dynamic view. Two main processes are described, presenting the main capabilities the system offers. On one hand, a detailed presentation of the message sending process is given. It shows how the components participate in order to offer end-to-end delay and confirmation support to the messages sent. On the other hand, the delivery report process is presented. Herein, the tracking of messages and their status is the main goal. It allows having knowledge at any moment about messages states.

3.6.2.1 Message Sending Process

As depicted in the following figure, this process is initiated by a sensor, a gateway or a base station when a new message arrives. Any of them registers a new message using *Registrar* interface. What this interface does is to put the message into a Queue waiting for *Listener* to be in charge of it. *Listener* is a daemon process, which is responsible for the surveillance of new messages that arrives. For this purpose, it executes asynchronous calls to *MessageQueue*. Once it discovers a message standing there, it takes the message and passes it to a new phase to be analyzed. This process is broken up into maximum 4 stages: analysis of destinations, *i.e.,* MTs, priorities, rules and throughout. These stages depend heavily on the environment where the middleware is deployed. Once the whole analysis completed, a *Sender* class is called to send the message. Later, the *ServiceLocator* class receives the *Sender* request in order to locate the service and the resource that will be responsible for disseminating the information towards the destinations. For such a purpose, this class takes into consideration basic information, such as MTs, priorities and rules. Once the resource is identified (*i.e.,* dissemination protocol with its parameters), *IRessource* begins to interact with the protocol, which finally is responsible to convey the information to the destinations, considering application requirements. At the same time, *ETEDM*, which is the process to offer timeliness support, is activated. It controls delay-

constraints for each message sent, while verifying the ACKs or NACKs sent by the protocol. If by the end of this time period no response is received, it asks the *ServiceLocator* to look for another service and resource to disseminate the information, *i.e.,* lookup process. This cycle is repeated based on the middleware configuration.

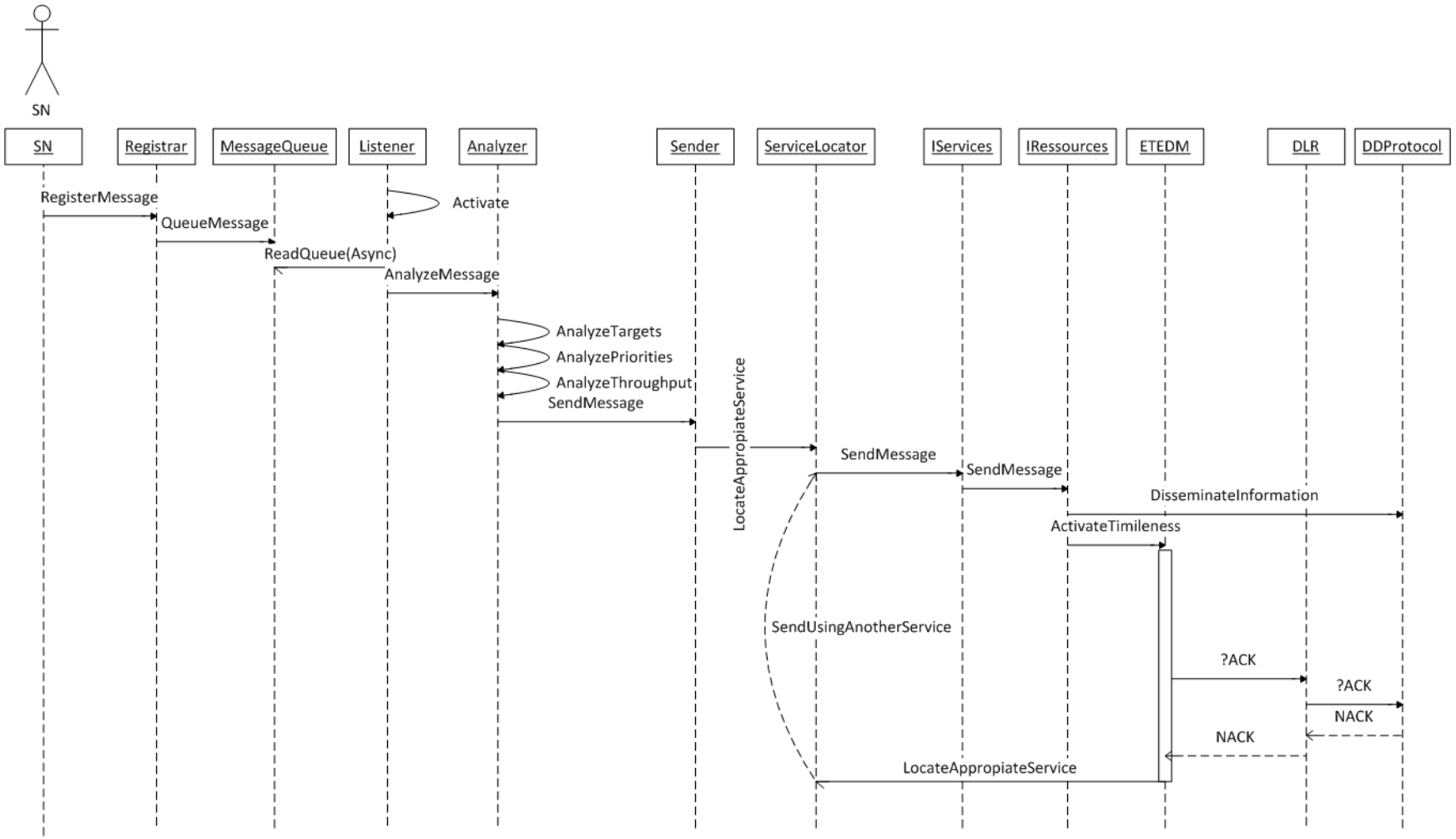

Figure 3-8 Message Sending Process

3.6.2.2 Delivery Report Process

Either another device, *i.e.*, sensor node, gateway or base station, or an internal component in the architecture, *e.g.*, Sender, might want to know at any time, the status of a sent message. The sequence shown in Figure 3-9 details how this process is executed. Once a device or component interrogates the *Status* interface, this request is transferred to the system and then, further analyzed to identity the message that is going to be tracked. Once this identification is performed, *ConfirmationAgent* is interrogated. It reads and analyzes the information presented by *EnvironmentRecorder*; which tracks all the events that happen with the message, such as end-to-end delay information, DLR and network failures. Based on this analysis, *ConfirmationAgent* presents a response to the system, which is sent back to the *Status* interface and then to the user o component interested in this information.

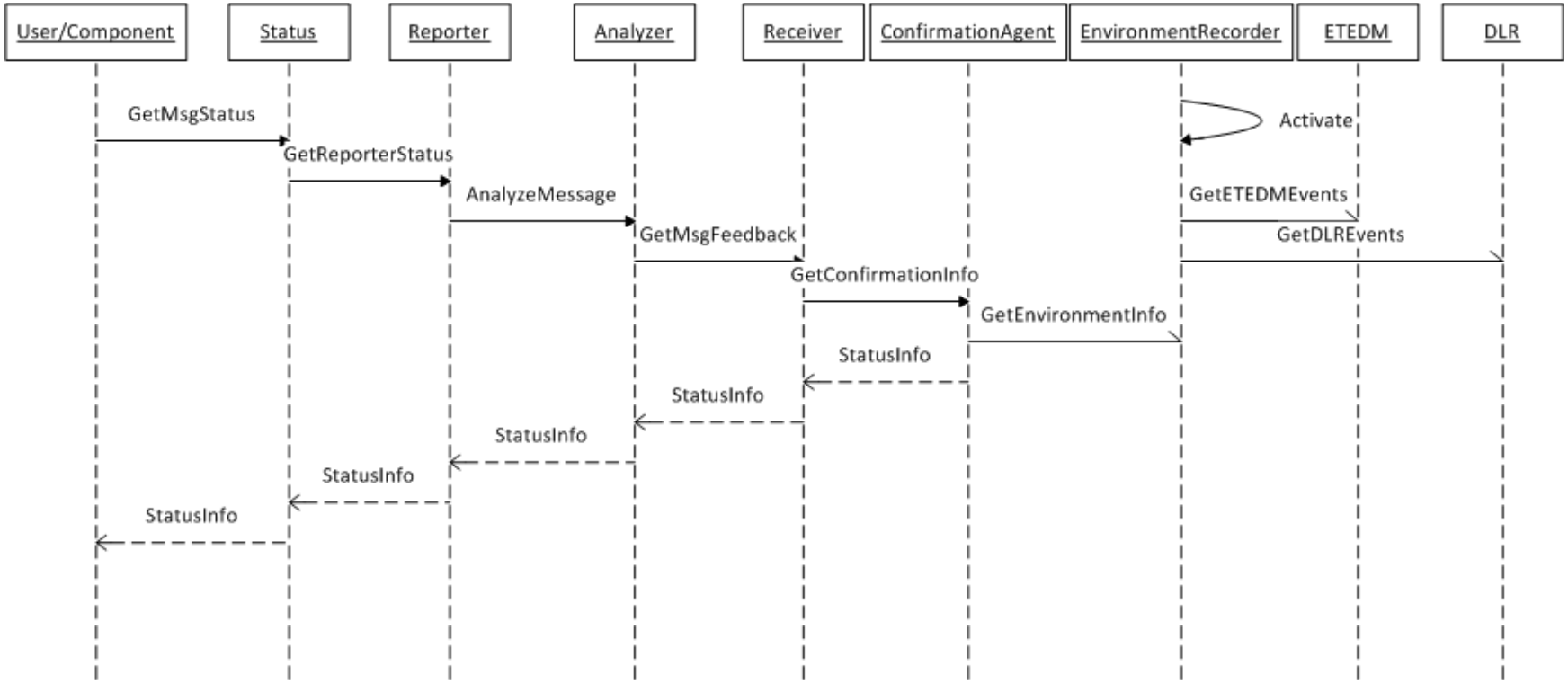

Figure 3-9 Delivery Report Process

3.7 **Middleware Editions**

As it was previously explained in section 3.1, the Application Engineering specifies the particularities of each of the product lines that are intended to be developed. Therefore, before defining each individual sub-process, it is required to identify each product line. In this thesis, the product lines will be known as *editions*. An *edition* is defined as a middleware component being executed on a particular device and network, whereas processing application requirements. Considering the roadmap previously shown in Figure 3-3, it collects the commonalities while keeping the configuration differences in each environment.

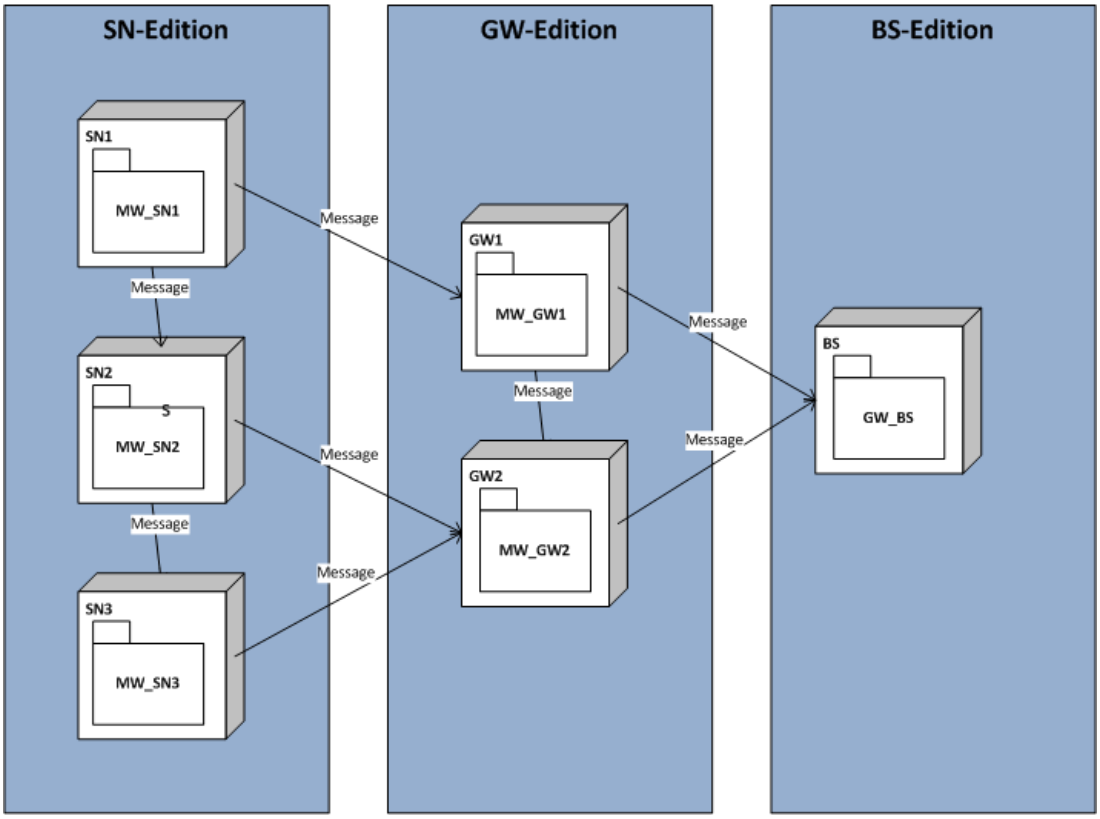

Figure 3-10 Middleware Editions

Three editions are identified in the middleware proposal and are depicted in Figure 3-10: *Sensor Node Edition* (*SN-Edition*), *Gateway Edition* (*GW-Edition*) and *Base Station Edition* (*BS-Edition*). As it can be inferred, these editions are taken from the device perspective presented in section 3.5.1; since the design should be adapted to each environment and respect its constraints. The following sections will further explain each edition and will follow the methodology for each

one of them. It is important to point out that the Application Design step will be omitted since the reference architecture will be used in all editions. The foremost changes will be mostly done in the Application Realisation phase.

3.8 SN-Edition

This edition is defined to be deployed in each sensor node inside the WSN. The challenges of this edition focus on trying to minimize energy consumption in the node while successfully executing the dissemination process.

3.8.1 Application Requirements Engineering

This category is in charge of the *SN-Edition* requirements definition in order to assure the basic needs of the system. It considers the Domain Requirements previously defined, but includes some additional functional requirements that are strictly related to dissemination of information inside the sensor nodes. The functional requirements concern energy efficiency, and can be stated as follows:

REQ 7. The dissemination process shall minimize energy consumption without compromising the maximal specified delay (*e.g.*, $t \leq 5$ sec).

REQ 8. The energy optimization shall be considered during all the process. The system shall avoid unnecessary retransmissions if not required (*i.e.*, number of retransmissions≅0).

3.8.2 Application Realisation

The application realisation is based on the *Reference Architecture*. This section analyzes the changes made to suit the specific edition. Figure 3-11 presents the class diagram for the middleware *SN-Edition*. The layers are basically the same; nonetheless, the inner components have been extended or modified to satisfy the edition requirements. The *Interfaces* component on top of the graphic relies solely on the *Registrar* class that allows the provisioning of the sending process. However, *Status* and *Administration* are not offered in this edition. The middle layer components have also been modified. First of all, the *Message Sender Manager* has also been specified. It consists of limited versions of the three main classes: *Listener*, *Analyzer* and *Sender*.

The *TargetAnalyzer* is not included in this edition since the user group information is in the base station. Then, the *Delivery Report Manager* is not present since the feedback to the application is not considered. The delivery report is directly obtained by interrogating DLR component. Finally, the *Data Access Manager* is partially implemented to allow access to the saved information and services. Nonetheless, this implementation does not consider administration capabilities. The third layer of the architecture, the *Service Manager*, responsible for interacting with the protocols and the network, offers a new class to support the energy-efficiency requirement specified for this edition. It keeps track on the energy consumption along the dissemination and delivery process to verify that it does not go over a specified limit.

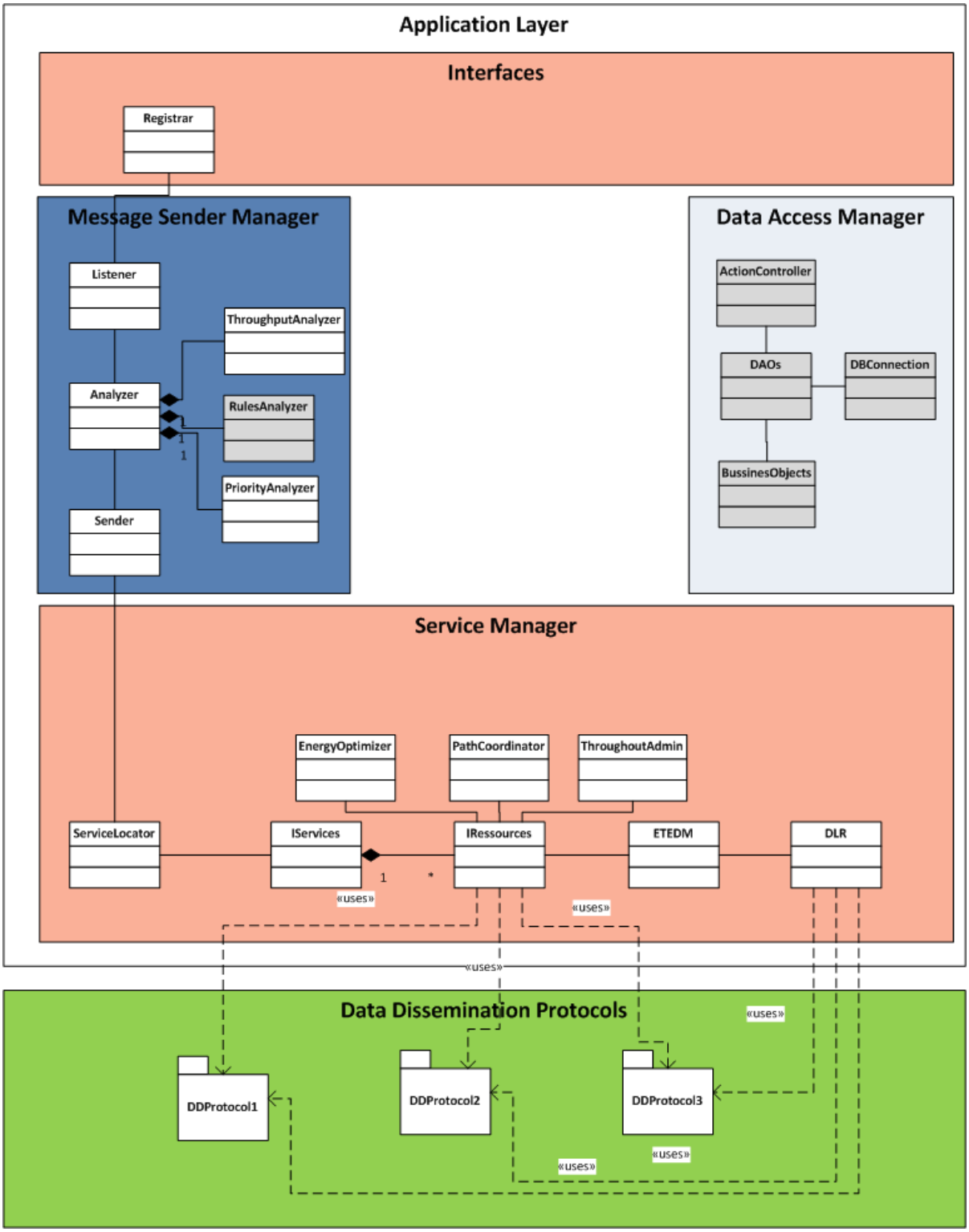

Figure 3-11 Class diagram of SN-Edition

3.9 **GW-Edition**

This edition is defined to be deployed in the gateway nodes inside the WSN. It is not as limited as the *SN-Edition*, since more powerful nodes are being used; nonetheless, it is not as complete as the *BS-Edition* which is fully functional. *GW-Edition* does not impose any special condition and only considers the *Domain Requirements* previously defined.

The application realisation is also based on the *Reference Architecture* but it includes some particular features. Figure 3-12 presents the class diagram for this edition. The *Interfaces* component presents *Registrar* and *Status* classes that enable the sending and feedback processes respectively. The *Administration* component is not offered in this edition. The middle layer components have also been specified in *Domain Realisation*. The *Message Sender Manager* is composed by specified versions of the three main classes: *Listener*, *Analyzer* and *Sender*. Neither the target, nor the rules analyzer are included since the user group information is in the BS and the rules should not be analyzed because all messages reaching this point should be strictly forwarded to the Base Station. Furthermore, the *Delivery Report Manager* is present with all its components since the feedback to the application is provided. The implementation of these components is limited to provide basic feedback and report to the *Status* class. Finally, the *Data Access Manager* is also partially implemented to allow access to the configuration information and services. However, it does not offer administration capabilities. The third layer of the architecture (*i.e.*, *Service Manager*) remains unmodified. Energy-efficiency is not considered in this edition since the gateways are considered more powerful devices than the sensor nodes and are normally plugged in to power sources.

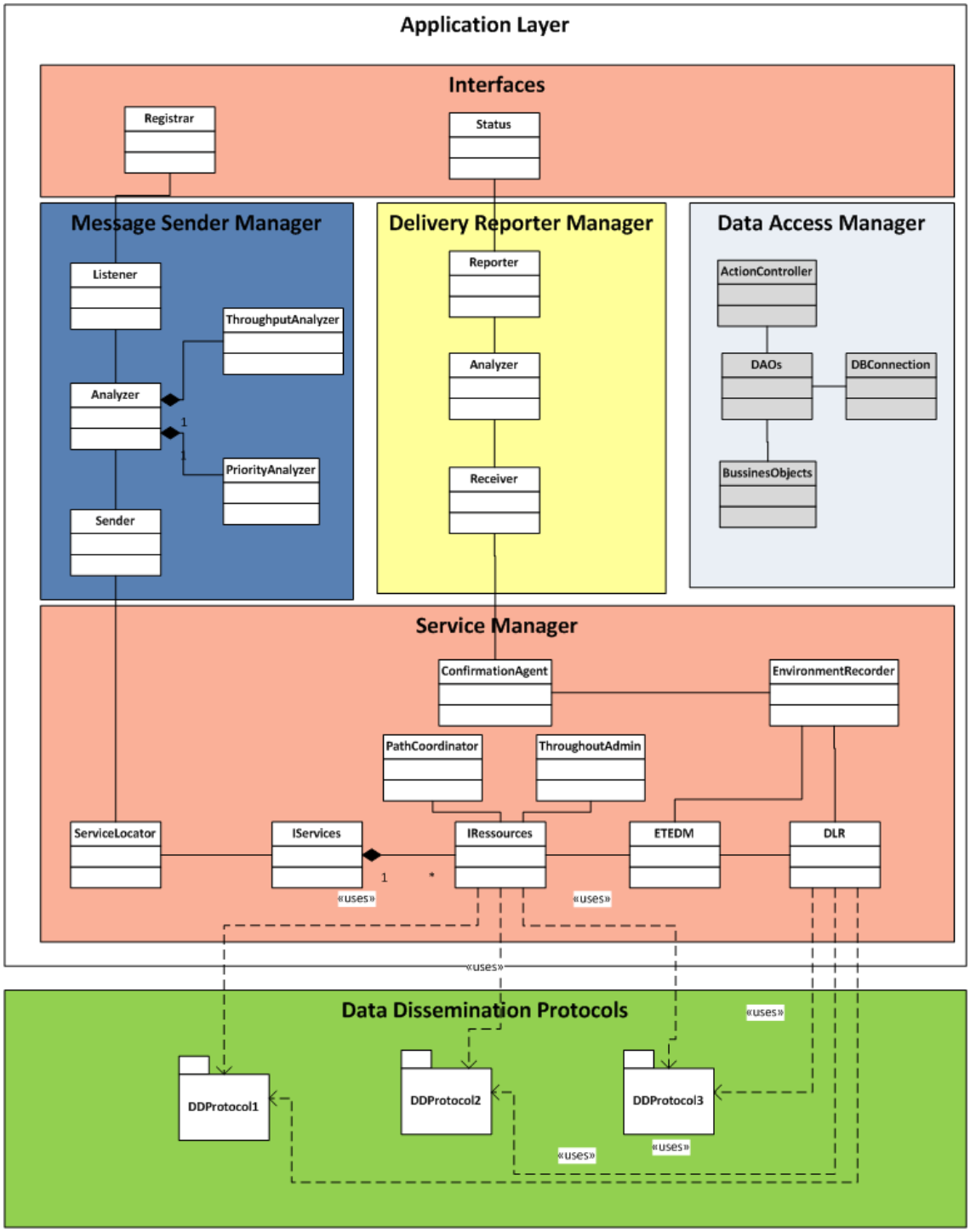

Figure 3-12 Class diagram of GW-Edition

3.10 **BS-Edition**

This last edition implements all features presented in the *Domain Engineering* process since this device is powerful enough to support them all and it is the entry point to other network integration. All components present in the architecture are fully implemented. Furthermore, this is the only edition in the system that will provide the *Administration* interface that allows users to manage the data model that includes creating, modifying or deleting users' information, priorities, rules, protocols and other business objects. Hence, the class diagram is omitted since it is the same as the one presented in Figure 3-7.

Chapter 4 PROOF OF CONCEPT AND RESULTS

In the previous chapter, the detailed architecture intended to meet the system requirements was presented. Now, this work is validated in order to guarantee that it certainly matches and achieves the requirements. For this purpose, it is important to use a proof of concept of a real scenario, where some measures are taken. A proof of concept is often used to demonstrate the feasibility of a system. It helps to understand the behavior of the components belonging to the architecture. The results should be, later on, evaluated to assure the completeness and consistency of the architecture. Therefore, the rest of this chapter is oriented to perform such evaluation. It is divided into four parts. The first section focuses on the scenario definition where the proof of concept is executed. The second section presents how the devices are deployed to perform our proof of concept. Later on, the middleware implementation is described in the third section. Finally, the fourth section takes the results obtained and proceeds with their analysis.

4.1 Scenario definition

A scenario is a useful tool to make a more realistic projection concerning the environment and the events where the system will be used. It helps to understand the variations that the system needs to address. In fact, each scenario presents different challenges, and forces the system to behave quite different. Understanding those variations will provide a better and more objective evaluation of the system. Therefore, in order to test and verify the feasibility of the proposed architecture, one scenario has been proposed, *i.e.*, light intensity goes below a threshold. This scenario is intended to disseminate information when the light intensity is lower than a certain value. Several sensor nodes deployed in the test-bed will be constantly sensing the light levels in specific areas. In the event, this value goes below a threshold; one or more sensors in the area will raise an emergency state. The sources will disseminate this information towards a gateway, which in turn will forward this information to a base station connected to Internet. User Groups (*e.g.*, Students, Professors) and Security Groups (*e.g.*, firefighters, rescue teams) are then notified. The message originator sensor, through its middleware, will coordinate the sending process. It must know if the message was correctly delivered and received. In such a case, it will receive a

positive ACK. Otherwise, a NACK produced by the forwarders upon failure, or a NR generated by itself if a timeout event is created, meaning that the message should be sent again by a different data dissemination protocol.

The main goal of this scenario is to obtain the measure of the end-to-end delay for a further analysis. For such a purpose, the proposed scenario is divided into seven steps as follows:

- Lights are initially on.
- After a random time, lights are turned off.
- Information, *i.e.,* light intensity, is periodically sent from the sensor nodes to the gateway.
- The gateway forwards the information to the base station.
- The base station analyses the data. If the measures sent by the sensor nodes go below than a pre-established value, *i.e.,* threshold, the middleware activates an alert.
- The middleware running on the base station looks for SG and UG members to notify them using the different protocols available, *i.e.,* SMS, email, twitter.
- The events, *e.g.,* end-to-end delay, are recorded for each message sent to SG and UG.

4.2 **Deployment of the devices**

Before describing the implementation environment, it is important to describe how devices are deployed. Herein, three types of nodes have been considered, as it was defined in the requirements and the architecture. As illustrated in Figure 4-1, the environment is made up of three sensor nodes, *i.e.,* MOs, one gateway and one base station. The middleware will be later deployed in some of these devices. Additionally, it considers the User and Security Groups where the notifications are finally addressed, *i.e.*, MTs.

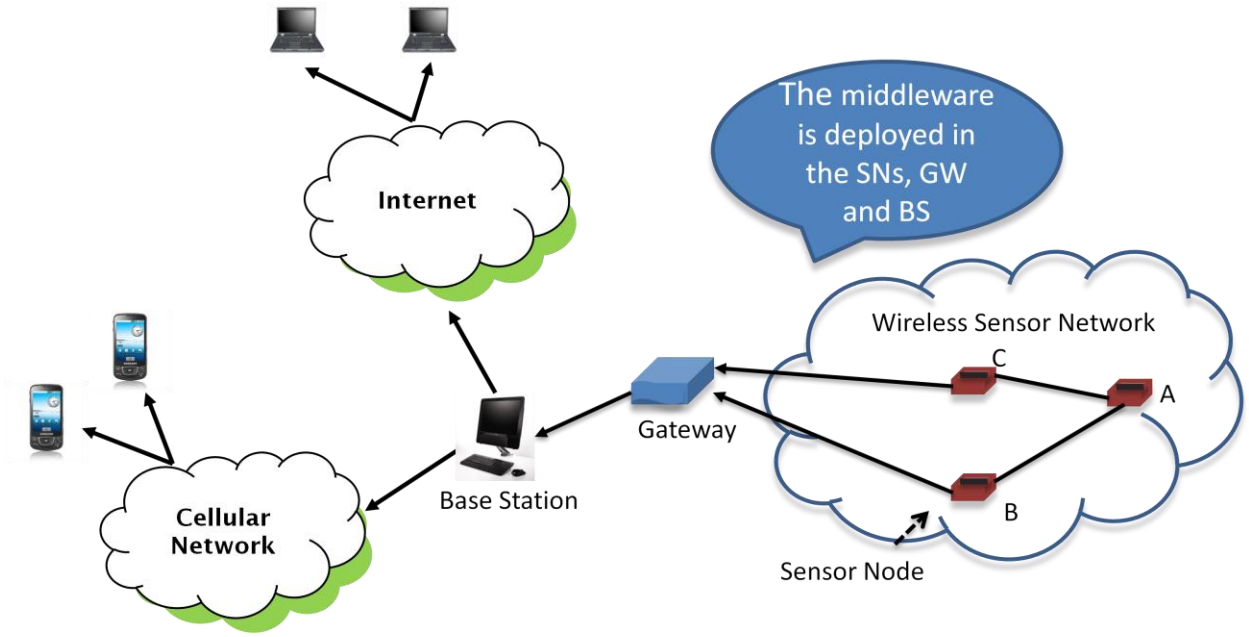

Figure 4-1 Prototype Deployment Environment

The WSN is composed by three nodes A, B and C that sense light intensity. These sensor nodes forward the sensed information towards the gateway which is connected to the base station. The technology used by the nodes is Crossbow motes MICA2 (MPR400CB). They are equipped with light and temperature sensors. They use Atmel ATmega128 microprocessors with 128 KB of flash memory and 4 KB of RAM. The radio communication for each MICA2 is at 916/433 MHz. Each device and the Gateway execute TinyOS Operating System (OS). It is an event-driving OS specially designed to run on resource-constrained network devices. The Programming Language (PL) to interact with TinyOS and its components is nesC. It embodies the structuring concepts and execution model of TinyOS. For the deployment and configuration of the WSN (*i.e.*, sensor nodes and gateway), MoteConfig 2.0 is used. Several values shall be taken into consideration for wireless communication purposes. The frequency of transmission is 916 MHz and the power transmission is -20 dBm, due to the small size of the test-bed. The gateway and the sensor nodes have the same group id (125), whereas the node id varies depending on each device. The gateway number is set to 0, while node numbers are set between 1 and 3. Figure 4-2 shows the settings previously described for Mica2 (*i.e.*, sensor nodes).

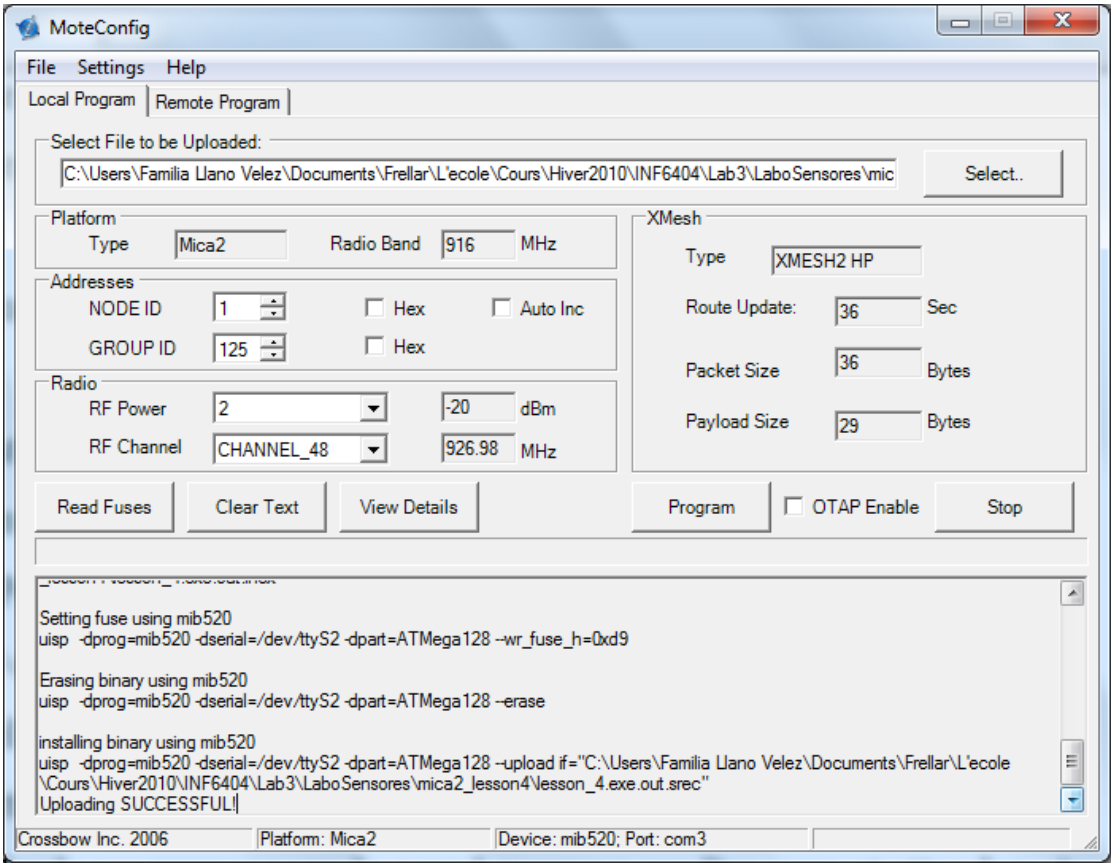

Figure 4-2 Mica2 settings

Once the configuration is finished, it is possible to convey information sent by each sensor node to the gateway and then to the base station.

Figure 4-3 presents in detail this information, as displayed in the base station. As it can be seen, it presents light information, *e.g.*, 2342 ADC mv; which is the behavioral value for the proof of concept. This information is the input required by the middleware to operate.

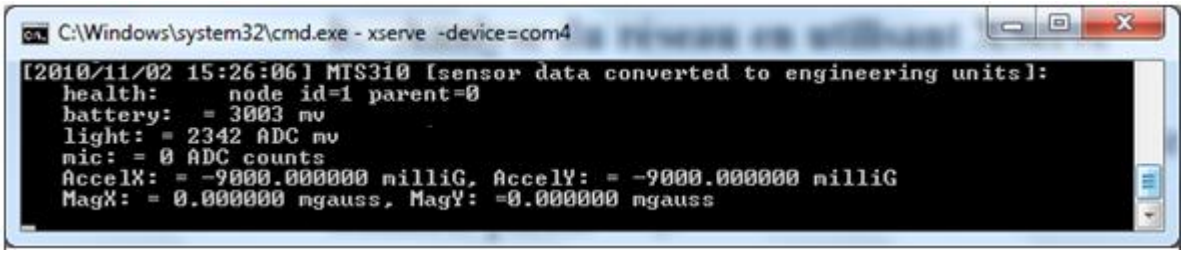

Figure 4-3 Data sent to gateway

Now that the sensor nodes and the gateway are configured, the next step is to set up the base station. For the proof of concept, this component is physically divided into two different

machines: *fixed-device* and *mobile*-device, as shown in Figure 4-4. The first one is connected to the WSN through the gateway and its main responsibility is to obtain the environmental information. Then, such information is forwarded to the *mobile-device*.

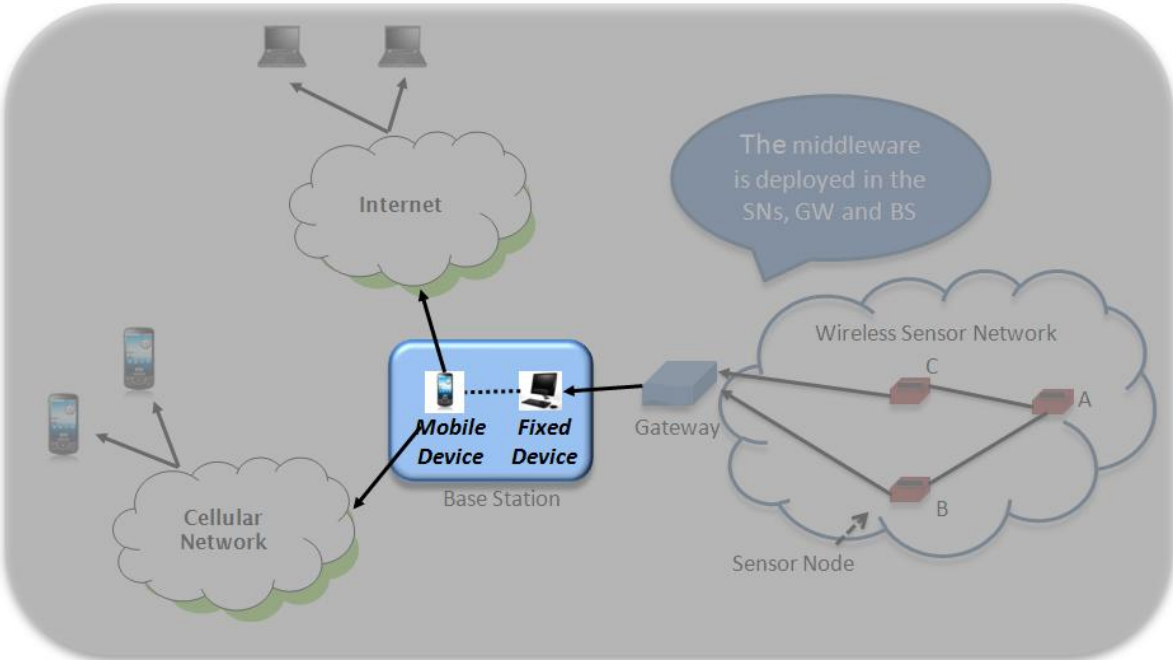

Figure 4-4 Base station close-up

Both devices have installed an instance of *announcer-application*, the screenshot of this application is shown in Figure 4-5. It is intended to capture WSN events at any moment. The first parameter of this application is *Light Threshold*. It permits the definition of the minimum value of the allowed light intensity. When a sensor sends a value lower than the minimum specified, the application calls the *middleware*. The second parameter is the *Group Id*. This value permits to filter the information, taking exclusively into account the nodes that belong to that group. The third parameter, *environment*, allows to establish the source of the information (*e.g.*, WSN). In order to read the WSN information, it provides a daemon that reads what each sensor node is sending (*e.g.*, light intensity). It can be activated by using *Init* button. In contrast, the process can be interrupted by pressing *Stop* button. Moreover, *Exit* button closes the application. Finally, the label *Listener* shows the status of the daemon process (*e.g.*, listening, stopped).

Figure 4-5 Announcer-application

The base station deployment, *i.e.*, using two physical devices, offers two advantages: Firstly, *versatility* is increased since one device is a wireless machine (*e.g.*, PDA). Thus, it could be moved to different locations, while keeping connection to WSN at any time. Secondly, *reliability* is also increased since the user could be notified using the PC, the mobile or both devices at a time. Both devices offer a configuration that guaranties the execution of the proof of concept. On one hand, the fixed-device is a Laptop equipped with 4 GB of RAM and Premium(R) Dual-Core CPU with 2.30 GHz executing 64-bit Windows 7 Home Premium as operating system. It has also installed Microsoft .Net Framework version 3.5 SP1 for middleware implementation purposes. On the other hand, a PDA acts as the mobile-device for the proof of concept. It has 64 MB of RAM and Marvell PXA310 624 MHz processor. For wireless communications, it is equipped with two interfaces: IEEE 802.11b/g Wi-Fi and Bluetooth 2.0 Bluetooth. It executes Windows Mobile as operating system. It has additionally installed

Microsoft .Net Compact Framework 3.5 to support the middleware services and Microsoft SQL CE for storage purposes.

4.3 Middleware implementation

As previously mentioned, once the *announcer-application* detects that the light goes below the threshold, the middleware begins its execution. The prototype focuses on implementing the *BS-Edition* for *fixed-device* and *mobile-device*. Its implementation is done by taking a subset of the main functionalities: *Interfaces, Message Sender Manager, Data Access Manager* and *Service Manager* discussed in section 3.10, meaning that not all classes are fully implemented. Only the features judged as critical to evaluate the feasibility of the proposed architecture are considered at this point. This decision is defined based on the impact each component has in the system requirements. The development of the *BS-Edition* is done under C# as a Programming Language (PL) using Microsoft .Net Compact Framework 3.5. The application is divided into three main logic components: *Interfaces*, *Business Rules* and *Data Services*, as shown in Figure 3-6. Each layer was implemented as an independent Component Object Model (COM) Project, which offers multiple advantages, such as portability, security, reusability and domain expertise encapsulation.

4.3.1 Interfaces Layer

The *Interfaces* layer exposes functionalities as services and variables. It provides a method called *registrar* for applications (*e.g., announcer-application*) to register events. The definition of the method is presented below:

```
public static void registrar(int priority,
                             String shortDescription,
                             String description,
                             String source,
                             int type,
                             String comments);
```

It receives six mandatory parameters. Initially, *priority* is used to establish the priority of the message (*e.g.*, 100=emergency). Then, *shortDescription* contains a brief description about the event (*e.g.,* light below the threshold). The third parameter, *description*, contains a more detailed description of the incident (*e.g.*, the sensor x registered a value of the light y in the z-building). Next, *source* indicates the origin of this information (*e.g.*, sensor node 1). Then, *type* refers to the type of the originator (*e.g.*, sensor node, gateway, base station). Finally, *comments* permits to include any additional information required to complement the message. This information can be presented in XML format for a better portability. Using this service, the events that come from the WSN (*e.g.,* light goes below a threshold) are initiated in the middleware.

4.3.2 Business Rules Layer

The *Business Rules* layer is the core of the system, since it implements the basic components: *Message Sender Manager*, *Service Manager* and *Delivery Report Manager*, enabling messages to be sent through different protocols, *e.g.*, SMS, Email and Twitter. To set up these protocols, a XML file is generated. It might be noticed that each protocol is composed by one or multiple resources, supporting the definition made in Figure 3-7. Table 4-1 describes the tags composing the file.

Table 4-1 XML resources file description

Tag Name	***Tag Description***
Protocols	Indicates the beginning and the end of the resources file
Protocol	Indicates the beginning or the end of a protocol
name (protocols)	Contains the name of a protocol
Classname	Describes the name of the class fully specified. *Package.ClassName.* This value is used by the middleware to dynamically execute the class using on-the-fly capabilities (*i.e.,* assemblies loaded and executed when needed). It allows the middleware to execute assemblies that might or might not be part of it.
description	Brief description of the protocol
Resource	Indicates the beginning or the end of a resource. For instance, a SMS could be sent using different SMS Gateways.
name (resource)	Contains the name of a resource.
param-name	Details all the parameters required to describe a resource. For instance, a SMS Gateway requires an IP address, a port, a user, a password and URL among others. It additionally describes maximum time to wait for a response and the probability of receiving an ACK.

As described in this table, each resource might require several parameter values to be described and configured. Figure 4-6 presents a fragment of the resources.xml file for the implemented prototype.

```
<?xml version="1.0" encoding="utf-8" ?>
- <Protocols xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  - <protocol>
    - <Protocol>
        <name>SMS</name>
        <classname>Announcer.Mobile.Services.SMS</classname>
        <description>Sends information using SMS</description>
      - <ressource>
        - <Ressource id="1" name="default">
          - <param>
            - <Param>
                <name>Originator.Address</name>
                <value>192.168.2.18</value>
              </Param>
            - <Param>
                <name>MaxTime</name>
                <value>45</value>
              </Param>
            - <Param>
                <name>Probability</name>
                <value>90</value>
              </Param>
```

Figure 4-6 Resources file

It can be noticed that the instance shows a SMS resource configuration. The tag *name* is used to identify the protocol used. The tag class describes the name of the class that implements the service. It is dynamically executed using on-the-fly .Net capabilities (also known as assemblies). This feature makes the environment execution more versatile, since it only requires setting up the XML. The information is sent in strict order according to its appearance in this file. The maximum set up time for each resource to complete its task is obtained from the XML file. This information is defined using a probabilistic approach based on studies done on the efficiency of these resources as stated by Pries *et al.* [15]. DLR interface is simulated using these probabilistic values to know whether the message was successfully received or not.

4.3.3 Data Services Layer

This layer is responsible for providing the interfaces to access the information. The information is mainly stored in two locations: database and XML resources file. Figure 4-7 presents the E/R (*i.e.*, Entity/Relation) diagram for the *BS-Edition*. It can be seen that there is a table called *queued_message*, where the message is initially queued using the *registrar* service. Then, the middleware, using the *listener* processes, moves the record to the *message* table. Later on, after the analysis is made, the single message is multiplexed into multiple records. Each message is addressed to a single user, using a different protocol and device (*e.g.*, SMS-blackberry, Email-iPhone), as defined in the XML file and in the database configuration. This information is stored in *sent_message* table. The DLR obtained from each service is recorded in

the *status* attribute. By using this information, the middleware knows the state of each single message sent to any user in the system.

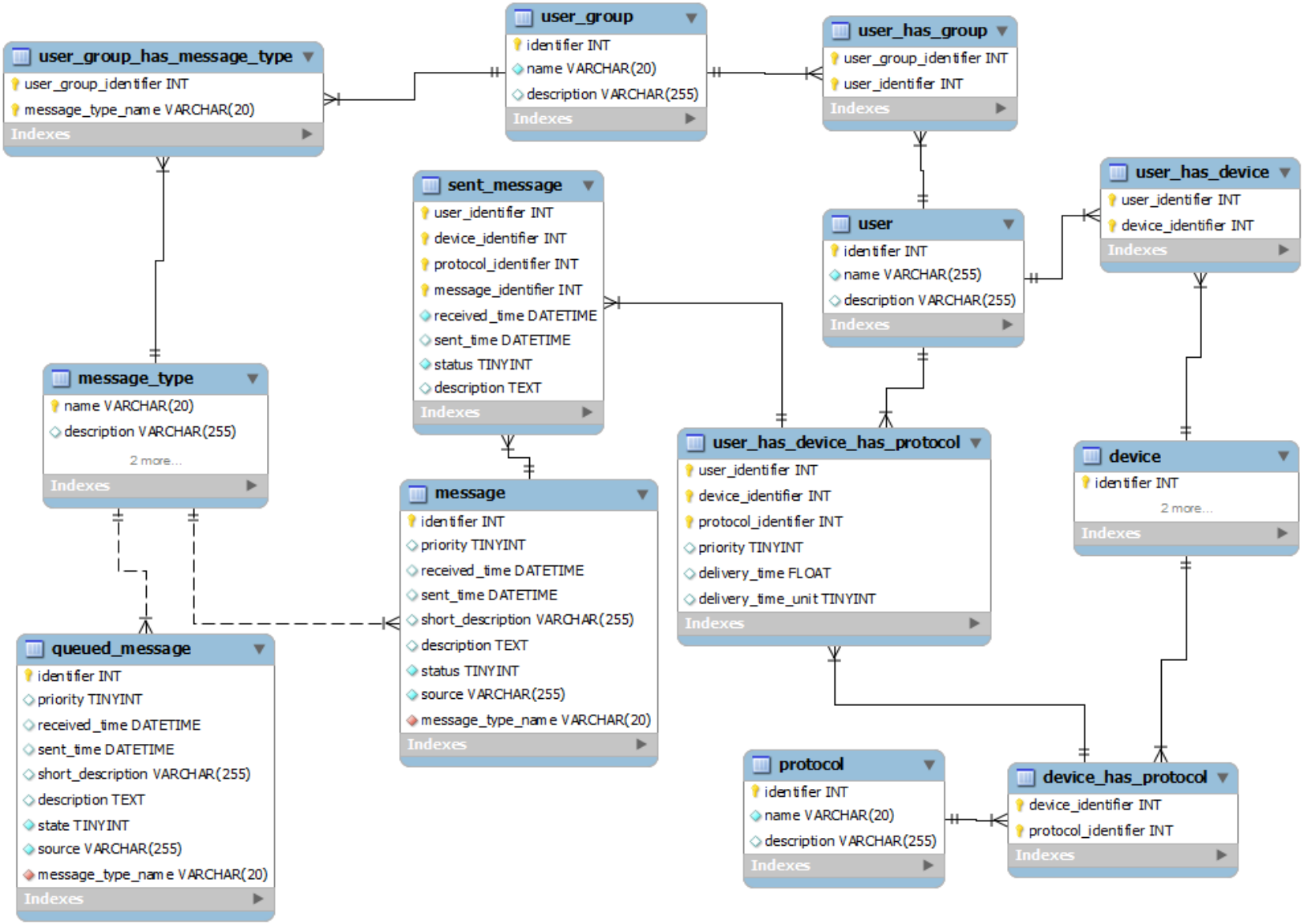

Figure 4-7 Entity/Relation Diagram

4.4 Analysis of end-to-end delay

Now that the environment has been deployed to perform the proof of concept, it is time to collect the results in order to make the analysis concerning the feasibility of the proposed architecture. The following subsections present how the results are taken and describe different analysis on such information made in MATLAB.

4.4.1 Results using mathematical approach

In order to verify the architecture, some tests were carried out using the proof of concept previously exposed. The results are analyzed using the mathematical approach for end-to-end delay evaluation presented in section 3.4. This approach defined three important equations. Firstly, the delay between the *originator* and the *terminator* with positive response was presented in equation (3.1). Secondly, the delay between the *originator* and the *terminator* with negative response was presented in equation (3.5). And finally, the delay between the *originator* and the *terminator* with no response was presented in equation (3.8). However, during the experiments, two variables: *middleware delay* and *lookup delay* were worthless when compared to the total delay; thus, they are not considered at this point. From equation (3.4), the resulting formula is:

$$\boldsymbol{delay_{ACK}} = (T_{i+j} - T_i) \quad \textbf{(4.1)}$$

where T_i is the time when the dissemination protocol is called by the middleware and T_{i+j} is the time when the middleware receives a response from the dissemination protocol.

Similarly, analyzing the equation (3.6), the following result is obtained:

$$\boldsymbol{delay_{NACK}} = (T_{i+j} - T_i) \quad \textbf{(4.2)}$$

Finally, considering equation (3.10), the result obtained is:

$$\boldsymbol{delay_{noresponse}} = \alpha_j \quad \textbf{(4.3)}$$

where α_j is a maximum time for a dissemination protocol to fulfill a *j*-task.

The deployment environment was depicted in Figure 4-1 and further explained in Figure 4-4 Base station close-up. For the proof of concept, a subset of the middleware BS-Edition (*i.e.* Base Station - Edition) was implemented. Each delay (4.1), (4.2) and (4.3) will calculated as the

elapsed time when sending a message between the base station, *i.e., fixed-device*, and the destination, *i.e.*, each device belonging either to the cellular network or to Internet.

For each message sent through a resource, *i.e.*, SMS, Email and Twitter, T_i is registered. Consequently, T_{i+j} is recorded after receiving an ACK or a NACK. With this information, $delay_{ACK}$ and $delay_{NACK}$ are produced, since they depend on these variables. Similarly, when a NR is received, α_j is recorded to produce $delay_{noresponse}$. Once all the information is collected for each message, the $delay_{end-to-end}$ (3.12) is calculated using these three individual delays. These values are then processed and analysed in MATLAB, in the light of the requirements verifying that these requirements are successfully met. Some statistical tables and graphics are presented to facilitate the analysis. The number of experiments performed is 20. In each experiment, 400 notifications are sent to the users. These numbers were motivated by [15], in order to have a more realistic approach concerning the maximum delay set up in the system.

Table 4-2 presents a fragment of the results obtained from the first experiment. The first column shows the corresponding statistical attributes analyzed: percentage of success, number of received ACKs, number of received NACKs or NRs, average delay and maximum delay for those successful messages (*i.e.*, messages with an ACK). For further details concerning all the experiments, refer to APPENDIX A. The percentage of success of the middleware for the first experiment is 98.25%. Accordingly, 7 destinations are not successfully notified among the 400 messages sent, corresponding to the remaining 1.75%. This will be further discussed in the following subsections. Another important conclusion that can be expressed is that the maximum end-to-end delay never exceeds the maximum time imposed to each resource. The highest delay of SMS with ACK is 45 sec, which is the specified limit. Concerning Email, the highest delay is 170 sec which is 5 less than the maximum permitted, *i.e.*, 175 sec. In the worst case scenario, the maximum end-to-end delay is defined as the maximum individual time for each resource (45 sec + 175 sec + 60 sec = 280 sec), as the middleware waits until the last moment to look up for the alternate resource (*e.g.*, it waits 45 sec when using SMS before choosing Email).

Table 4-2 Results from the first experiment

	SMS	*Email*	*Twitter*	*Middleware*
Percentage of Success	79.50%	70.73%	70.83%	98.25%
ACK	318	58	17	393
NACK, NR	82	24	7	7
Average Delay (sec)	24.94	99.72	30.17	154.83
Maximum Delay (sec)	45	170	59	280

To examine these results in more detail, MATLAB is used; particularly, its statistic and its graphic capabilities. Two types of examination are performed. On one hand, a random subset of individual messages is taken in order to make a close up to the information produced. On the other hand, all the information for each experiment is taken into consideration, performing statistical analysis on the whole set of data.

4.4.2 Analysis on individual messages

To perform the first analysis, 12 random messages were selected from the first experiment. Each individual message was successfully received by the destination. Table 4-3 presents delay and status information, *i.e.*, NR, NACK or ACK, associated to each protocol. The first column shows the message id. The following three columns, which are split into two inner columns, detail the protocol end-to-end delay and partial status for SMS, Email and Twitter. The last column presents the final status of a message, which shows whether the message was successfully received or not. For instance, for the fourth message, *i.e.*, message id = 4, the system sends the message by SMS, but after exceeding its threshold, *i.e.*, 45 sec, the partial status is set to NR; then, the system sends the same message by email; however after 73 sec, the system receives a NACK from the protocol; therefore, another resource is required to convey the information, *i.e.*, Twitter, which succeeds to disseminate the information towards the destination. The response of the successful complexion of the task is given by Twitter to the middleware after 17 sec. In this case, the message takes 135 sec, *i.e.*, 45 sec + 73 sec + 17 sec, to reach the destination (including the confirmation). Furthermore, Figure 4-8 illustrates the results. It helps to

see how the middleware makes its decisions based on the end-to-end delay. The vertical lines, *i.e.*, Y = 45 sec, Y = 60 sec, Y = 175 sec, help the user to better visualize the maximum delay set to each protocol. Sometimes, the message is successfully sent, using all three protocols, *i.e.*, messages 1 to 4. Other times, only a subset of the available protocols instead of the whole set is required, *i.e.*, messages 5 to 12. Indeed, those messages using a subset of the protocols are likely to have lower delays.

Table 4-3 Results of twelve individual messages

Message Id	***SMS***		***Email***		***Twitter***		***Final Status***
	End-to-end delay (sec)	*Partial status*	*End-to-end delay (sec)*	*Partial status*	*End-to-end delay (sec)*	*Partial status*	
1	45	NR	161	NACK	36	ACK	ACK
2	45	NR	79	NACK	59	ACK	ACK
3	45	NR	85	NACK	59	ACK	ACK
4	45	NR	73	NACK	17	ACK	ACK
5	8	NACK	155	ACK			ACK
6	10	NACK	164	ACK			ACK
7	45	NR	109	ACK			ACK
8	33	NACK	62	ACK			ACK
9	37	ACK					ACK
10	39	ACK					ACK
11	15	ACK					ACK
12	41	ACK					ACK

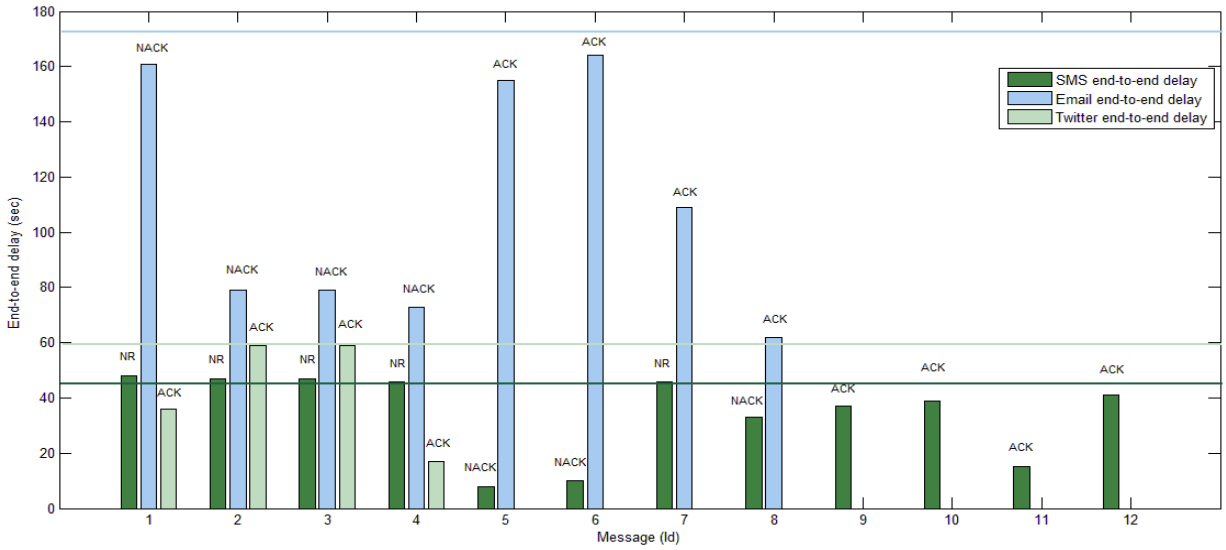

Figure 4-8 Middleware decisions based on messages status

4.4.3 Analysis on experiments

While the previous sub-section focuses on the analysis on particular messages, this part focuses on making an overall study over all sent messages, *i.e.*, the 20 experiments carried out. Herein, three analyses are made: maximum end-to-end delay, average end-to-end delay and percentage of success. Each of these examinations reveals different aspects of the middleware.

4.4.3.1 Maximum end-to-end delay analysis

As explained in 4.4.1, the end-to-end delay for the proof of concept is the summation of the individual delays (4.1), (4.2) and (4.3), from the base station to a device either in the cellular network or in Internet. Those end-to-end delay results are used at this point to perform the maximum end-to-end analysis which can be defined as the highest delay registered by the middleware to send a message and to receive a response in that context. Therein, the maximum end-to-end delay registered by each resource in each experiment is presented. Figure 4-9 divulges that none of the delay values exceeds its limits. For instance, the maximum SMS delay by each experiment is lower or equal to 45 sec. The same scenario applies for Email and Twitter. It demonstrates that the middleware takes on-time decisions based on the defined thresholds, *i.e.*, 45 sec for SMS, 175 sec for Email and 60 sec for Twitter.

Another important analysis that can be done is that Email service takes more time than Twitter to send the information in the majority of the experiments. This might be due to several factors, *e.g.*, efficiency and reliability, which are out of the scope of this research. However, the relevance of this observation is that, based on this information, one can make decisions regarding the configuration of the resources in the XML file, in order to change priority and execution order. For instance, if a resource, *e.g.*, Email, takes much more time to notify a person compared to another, *e.g.*, Twitter, and both offer similar percentage of success, then Twitter should appear before Email in the XML file. Accordingly, a wide set of users might be notified with less delay, thereby, maximizing the effectiveness of the whole system.

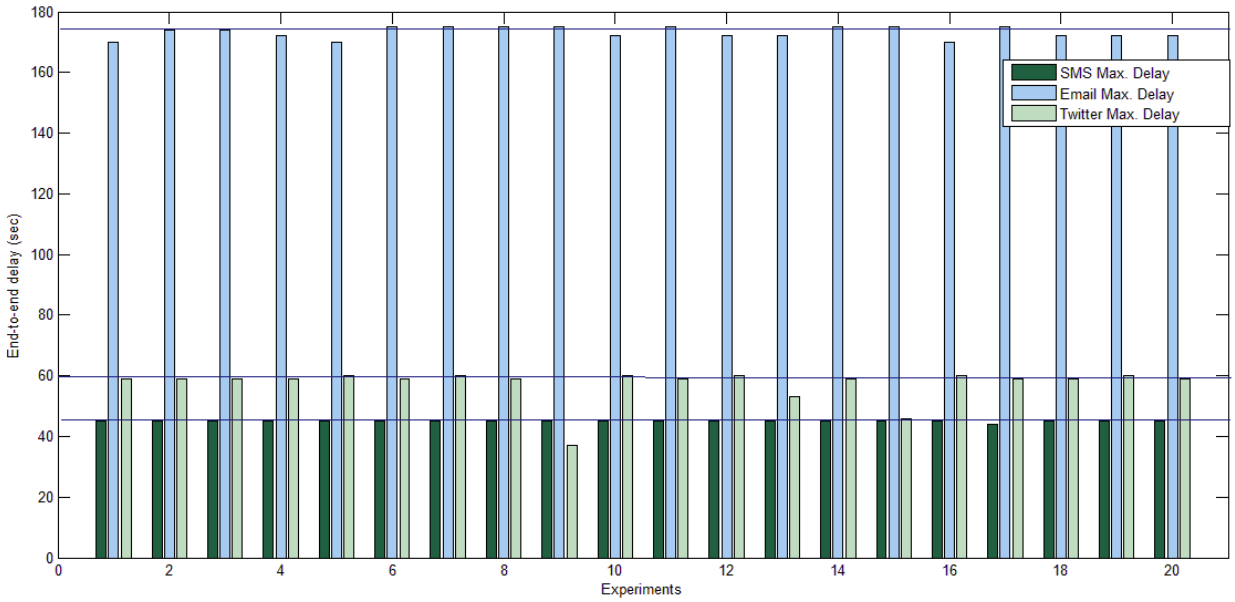

Figure 4-9 Maximum end-to-end delay analysis

4.4.3.2 Average end-to-end delay analysis

Similar to 4.4.3.1, this analysis consider the end-to-end delay results taken in the proof of concept. The average end-to-end delay can be defined as a common delay registered by the middleware to send a message and receive a response. Since the statistical average gives information about tendencies; analyzing it, helps to better understand the results. For this purpose, Figure 4-10 is presented, which can help to make important decisions such as configuration adjustments. Efficiency, for instance, can be improved. Herein, a reduction in the

maximum delay time parameter for the resources in the XML file could be performed without a significant impact. It is recommended to perform several tests when adjusting this parameter in real environments, to avoid unexpected results. Particularly, analyzing the figure below, it can be inferred that end-to-end delay for Email in average never reaches 120 sec; therefore, this would be a candidate resource, whose maximum delay time, *i.e.*, 175 sec could be reduced. Accordingly, this adjustment would affect the whole behavior of the system reducing the end-to-end delay for user. For instance, if this parameter is reduced to 130 sec the maximum end-to-end delay would be 235 sec instead of 280 sec, *i.e.*, the current maximum end-to-end delay.

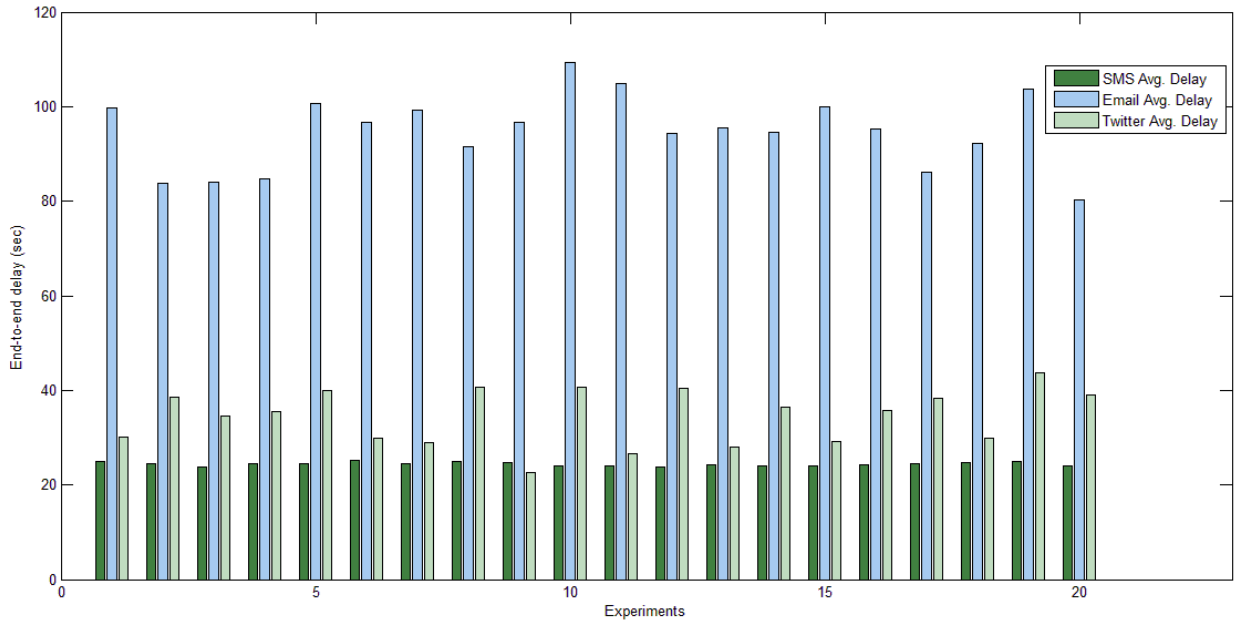

Figure 4-10 Average end-to-end delay analysis

4.4.3.3 Percentage of success

A final analysis to the information concerns the percentage of success, *i.e.*, *R(t)*. It is defined as the relation between the number of messages sent and those successfully received by the system. Considering M as a message sent from the originator to the terminator using the middleware, $M(t)_{ACK}$ is a function that represents the successful or unsuccessful reception of a message in t time-units. It can be expressed as follows:

$$M(t)_{ACK} = \begin{cases} 1, if\ M\ is\ successfully\ received\ by\ the\ terminator\ in\ t\ time-units \\ 0, otherwise \end{cases} \quad \textbf{(4.4)}$$

Then, the percentage of success of the system can be represented as follows:

$$R(t) = \left(\frac{1}{n}\sum_{K=1}^{n} M(t)_{ACK}\right) * 100\ \% \quad \textbf{(4.5)}$$

Where $M(t)_{ACK}$is given by equation (4.4) and n is the total number of messages sent to the destinations.

Now, the equation (4.5) is used to analyze each resource, *i.e.*, SMS, Email and Twitter, in each one of the 20 experiments. The middleware is additionally evaluated using this equation by considering all messages sent by the different resources. Figure 4-11 shows that the middleware outperforms the success offered by these resources individually.

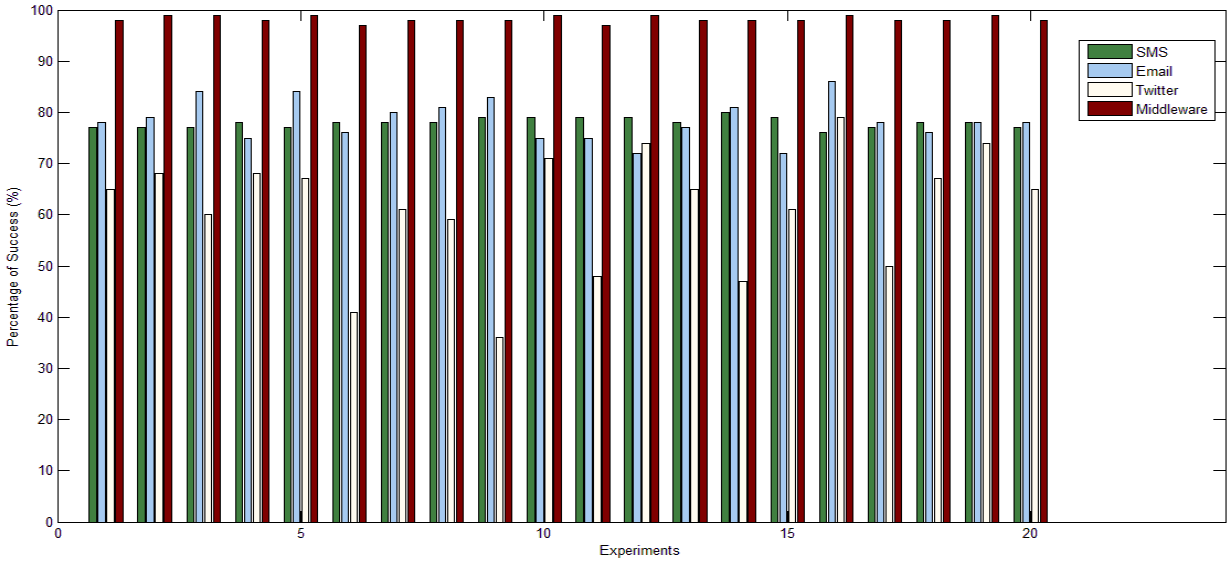

Figure 4-11 Comparing percentage of success by resource

It can be seen that the overall success of the middleware, *i.e.*, a message reaches the user in a maximum time defined, in average is close to 98%. This represents a great improvement when compared with the performance of the resources individually. SMS, for instance, shows an average success of 78%. A slightly increment is seen in Email with 79%. Finally, Twitter, in turn, offers the lowest success of the three individual resources (61%). The success of the middleware is given mainly due to the proactive approach employed in the end-to-end delay. The right configuration of the maximum time values for each resource is another key element since it maximizes the ACK, while minimizing the individual delays required by SMS, Email and Twitter.

Chapter 5 **CONCLUSION**

After presenting a delay-constrained middleware-architecture proposal for disseminating information in WSNs that considers QoS constraints and validating this architecture using a proof of concept of a real scenario, where some measures were taken to analyze and demonstrate the feasibility of the system, we can now conclude the thesis. In this conclusion, the work done is divided as follows. The first section presents a summary of the proposal and its results. The second section details the limitations of this work based on some assumptions. Finally, the last section proposes the future works that could be done in this research area.

5.1 **Summary of the work**

In the last decade, the development and deployment of WSNs has considerably increased [1-3]. Their main goal is to gather environmental information and then transmit it to a destination either inside the WSN or in another network (*e.g.*, Internet or Cellular Network) [2-5]. For this purpose a data dissemination protocol is normally used. Delay-constrained applications in this context usually impose QoS constraints, *e.g.*, end-to-end delay, to prevent accidents and coordinate rescue operations [5]. Therefore, a middleware acting as a mediator between delay-constrained applications and data dissemination protocols is required. Using these protocols, our solution could convey information towards the destinations, and when a maximum end-to-end delay constraint is exceeded, the middleware could make on-time decisions, *i.e.*, send the information by using another data dissemination protocol. When disseminating information two major issues need to be analyzed. On one hand, a single protocol or technique is used to convey the information from the source to the destination, even if some of them offer QoS parameters [2, 8]. It creates a complete dependency on the data dissemination protocol. On the other hand, those protocols or techniques are designed to be executed on a specific environment [2-5], (*e.g.*, a WSN or Internet). These characteristics impose serious shortcomings when disseminating delay-constrained information among several networks. Accordingly, the existing techniques do not guarantee conveying the information from the source to the destination. Therefore, the main goal of this research is to design and develop a middleware architecture that enables the dissemination of information in WSNs considering QoS constraints.

Before presenting our solution, the state of the art is analyzed focusing mainly on data dissemination techniques, and middlewares/frameworks proposed by the research community. This literature review is done in the light of some parameters. Firstly, the parameters used for data dissemination techniques are: *end-to-end delay*, *energy efficiency*, *transmission rate*, *confirmation mechanisms*, *congestion control* and *percentage of success*. In this analysis eight data dissemination protocols are considered: *CCBs*, *DD*, *TTDD*, *GRAB*, *FDDDP*, *DDDP*, *CBDDP* and *EAGDDP*, (presented in Table 2-1). From this analysis, some important characteristics are uncovered: 1) no protocol fully considers the general parameters, therefore, the middleware should offer mechanisms their completeness 2) understanding the protocols helps the middleware design to improve the data dissemination process. Additionally, other parameters are used to evaluate the existing middleware and framework proposals: *general purpose*, *transparency*, *adaptive* and *extensible*. In this case six architectures (*i.e.,* middlewares and frameworks) are reviewed: *Cygnus*, *Chameleon*, *VANET-Middleware*, *MILCO*, *SensorBus*, and *Disaster Management Framework* (presented in Table 2-2). From this evaluation *SensorBus* and *VANET-Middleware* reveal important characteristics that should be taken into consideration when designing our middleware architecture.

Now that the literature review has been explained and in order to design the architecture, it is important to understand the end-to-end delay using a mathematical approach, since it is one of the most important requirements of this research. To perform this analysis Figure 1-2 was presented; it shows the process to convey the information from A to F when several forwarders are required. In each segment there is an *originator* and a *terminator* and in between there is a delay. In this context three scenarios may happen. 1) The *originator* could receive a positive response from the *terminator* in a valid period of time, *i.e.,* ACK, 2) the *originator* could receive a negative response in a valid period of time, *i.e.,* NACK or finally, 3) the *originator* could not receive any response from the *terminator, i.e.,* NR. In each scenario, the delay is completely different. Each one of these scenarios and the end-to-end delay are now analyzed. Equations (3.4), (3.7), (3.10) and (3.12) present a mathematical approach to describe these scenarios. Now that the delays are analyzed, the design can be created. The delay-constrained middleware for disseminating information in WSN is therefore presented as three-layer and three-plane reference architecture (presented in Figure 3-6). On one hand, the layers are *interfaces*, *business rules* and *data services*. On the other hand, the three planes acting upon the three layers are *QoS*,

Confirmation and *Timeliness*. The *interface* layer is exposed as a standard-access service that exposes a unique way for MO to register alerts, to provide message status information and to manage the data model. The *business rules* layer is the system core functionality, whose main responsibility is to guarantee the message sending process and the delivery report process. For this purpose, it manages the communication with the enabled data dissemination protocols. Finally, the *data services* layer offers the functionalities to access data information, *e.g.*, data protocols, priorities and destinations. The planes are oriented to offer end-to-end delay, confirmation and timeliness support through the presence of daemon processes that are executed continuously. The middleware is intended to be deployed in three main devices: the sensor nodes, the gateway and the base station. Accordingly, the edition concept is introduced in this work. An edition, thus, is understood as a middleware component executed on a particular device. It collects the commonalities whereas keeping the configuration differences in each environment. The *SN-Edition* was defined to be deployed in each sensor node. The *GW-Edition* was designed to be deployed in the gateway nodes. Finally, the *BS-Edition* is deployed in the base station.

Now that the architecture has been presented, a proof of concept of a real scenario is implemented in order to evaluate the feasibility of the system. In this scenario, sensor nodes forward light intensity information towards the destinations, (*i.e.*, users in cellular networks and in Internet), once it goes below a pre-established threshold. The deployment environment for the prototype to be implemented is determined. Since the proof of concept aims to evaluate the feasibility of our proposition, a subset of the whole architecture is considered. Since the *BS-Edition* includes the main functionalities defined in the architecture this edition has been considered to be implemented in this proof of concept. It is physically divided into two different machines: *fixed-device* and *mobile-device*. These devices receive the information sent by the sensor nodes and take the decision to disseminate towards the destinations. Now that the proof of concept is defined, it is time to take the results in order to perform the analysis. But before that comes, the mathematical approach for end-to-end delay previously discussed is further analyzed. Since *middleware delay* and *lookup delay* are worthless, the resulting equations are: delay with ACK (4.1), delay with NACK (4.2) and delay with no response (4.3). To perform the proof, 20 experiments were carried out, sending 400 notifications to the users in each one. Three protocols are used to notify the users, *i.e.*, SMS, Email and Twitter. Each individual delay is recorded. Based on this information, two different analyses are performed: analysis on individual messages

and analysis on experiments. For the first analysis, 12 random messages were selected. It allows seeing the middleware decisions based on the end-to-end delay and the confirmation status registered by each protocol. For the second examination, three perspectives are used: maximum end-to-end delay, average end-to-end delay and percentage of success. The maximum delay examination reveals that never a delay exceeds its threshold. This observation also helps as a guide to change the order of the data dissemination protocols based on the percentage of success comparison between two protocols. Furthermore, the average end-to-end delay analysis shows how efficiency can also be improved. A maximum delay for a protocol can be adjusted based on observations. As it was witnessed in the experiments, Email maximum delay for example could be reduced from 175 sec to 120 sec. Finally, the percentage of success is also analyzed. It reveals a middleware success close to 98%, which is highly superior to the success of the individual resources, such as SMS (*i.e.,* 78%), Email (*i.e.,* 79%) and Twitter (*i.e.,* 61%).

These results show how data dissemination requirements are successfully fulfilled. On one hand, end-to-end delay and percentage of success are achieved as shown in the analysis of the notifications. End-to-end delay is achieved by using a pre-established maximum period of time for each message sent to a destination through a resource. If the message is not successfully received during this interval, the system searches another resource to send the information. The percentage of success is met by the architecture with the inclusion of several resources, *i.e.*, SMS, Email and Twitter, which increase the percentage of successfully delivering a message. From this analysis, it can also be concluded that energy efficiency, transmission rate, confirmation mechanisms, control congestion are also fulfilled. Energy efficiency strategies are achieved by the architecture's ability to make decisions to disseminate or not some particular information. For such a purpose, it classifies the information depending on groups established in the WSN. If the middleware receives information not belonging to a certain group, it then ignores it. Consequently, the information is not longer disseminated. As a result, energy is saved in the subsequent sensor nodes; contributing to reduce energy consumption in the whole WSN. Moreover, transmission rate and confirmation mechanisms are also considered and met by this proposal. Using the delivery report (DLR) component, the architecture knows what happens to each message at any moment; thus, offering confirmation mechanisms depending on the message status, *i.e.,* ACK, NACK, and NR. This component also gives the possibility for the system to use more than one resource at the same time in order to make the notification an efficient process,

guaranteeing the transmission rate requirement. Control congestion is also achieved through the DLR, which provides the message status. Since this status is known in advance, unnecessary retransmissions are avoided, helping to prevent congestion in the network. The architecture requirements are also met. The architecture is considered for general-purpose, since it is totally independent from the applications and the underlying data dissemination protocols used to convey the information towards the destinations. Additionally, the use of an interface-oriented approach, *e.g.*, service layer, to expose its services, gives the middleware transparency. Applications using the middleware require only the interface definition to interact with it. Furthermore, in order to make the middleware adaptive, certain parameters, such as maximum time are adaptable. It gives the system the possibility to dynamically define, for instance, the time constraints for each resource used to disseminate the information. It also allows defining destinations to be notified and message priorities. Finally, the architecture is extensible, since its design considers the future inclusion of new functionalities such as data dissemination protocols.

5.2 **Limitations of the work**

The work that has been proposed presents several limitations that should be taken into consideration when using the proposal and when defining a future research path. Initially, the middleware heavily relies on a data dissemination protocol to provide the status of the disseminated information. Accordingly, the middleware offers an interface which can be used by applications to know this status. If the used protocol does not offer such information, it automatically becomes useless in the architecture. Secondly, in order to save energy in the WSN, the protocol must offer a way to express these types of requirements, *e.g.*, express Degree of Interest (DoI) as is done by CCBs [2]. Additionally, when conveying information from WSN to cellular networks, the presence of a gateway towards the cellular network, *i.e.*, an interface to ask for the messages status, is assumed. For instance, Kannel [16] is a SMS gateway that offers such capability. Therefore, when there is no gateway, the architecture cannot guarantee a successful delivery with delay constraints in cellular networks. Finally, in this architecture, we assumed that each device that receives a message has a way to send back a response about a successfully or unsuccessfully reception. If this functionality is missing in such device, the delivery cannot be guaranteed.

5.3 **Future Works**

Based on the proof of concept here developed/evaluated and the presented limitations, some future works are proposed. As known from the proof of concept, the implemented prototype is BS-Edition, which runs on a base station where the delay responses of the data dissemination protocols are simulated. A more realist scenario should include two improvements. Firstly, the full implementation of the three editions, *i.e.*, SN-Edition, GW-Edition and BS-Edition, should be performed. Secondly, the middleware should work with real data dissemination protocols instead of using a probabilistic approach to simulate the delay and the confirmations. Furthermore, the dissemination should be done in a real situation, *e.g.*, simulate emergency evacuation.

Once these improvements have been done, there is another important work to be executed. Since the middleware editions are strictly conceived for WSNs, in order to obtain more reliable results, it is imperative to include new editions to be used in other networks, *i.e.*, Internet and cellular networks. For instance, there should be an Internet edition to work in each user device, *e.g.*, laptop, PC, smart phone. Similarly, an edition that runs on cellular phones could be also included. All of these oriented to reinforce the confirmation mechanisms, as it is an important requirement of the system.

A final future work identified in this research is the extensibility of the middleware to disseminate information to other networks. Indeed, this middleware was designed to disseminate information from WSN to Internet and cellular networks. A further extension is quite interesting. Since a lot of research is currently being done in the vehicular domain and foreseeing that a lot of people is going to spend a lot of time in their vehicles, the development and deployment of an edition for VANETs could produce interesting results. It will require making an in-deep analysis of its architectures, *e.g.*, Wireless Access in Vehicular Environments (WAVE), and protocols. Furthermore, it could also be important to analyze wider extensions including fourth generation (4G) networks such as Worldwide Interoperability for Microwave Access (WIMAX) 2.0, using protocols that might meet IMT-Advanced requirements such as 802.16 and 802.21.

REFERENCES

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, pp. 2292-2330, August 2008.

[2] H. M. Ammari and S. K. Das, "A trade-off between energy and delay in data dissemination for wireless sensor networks using transmission range slicing," *Computer Communications*, vol. 31, pp. 1687-1704, June 2008.

[3] D. Virmani and S. Jain, "Comparison of proposed data dissemination protocols for sensor networks using J-Sim," in *2009 IEEE International Advance Computing Conference. IACC 2009*, Patiala, India, 2009, pp. 1179-1186.

[4] Y. Zhang and L. Wang, "A comparative performance analysis of data dissemination protocols in wireless sensor networks," in *Proceedings of the 7th World Congress on Intelligent Control and Automation*, Chongqing, China, 2008 pp. 6663-6668.

[5] S. Saha and M. Matsumoto, "A framework for disaster management system and WSN protocol for rescue operation," in *TENCON 2007 - 2007 IEEE Region 10 Conference*, Taipei, Taiwan, 2007, pp. 1315-1318.

[6] Y. Faheem, S. Boudjit, and K. Chen, "Data Dissemination Strategies in Mobile Sink Wireless Sensor Networks: A Survey," presented at the 2009 2nd IFIP Wireless Days (WD 2009), Paris, France, 2009.

[7] A. R. L. Ribeiro, L. C. Freitas, C. R. L. Frances, and J. C. W. A. Costa, "Middleware performance evaluation in wireless sensor networks," in *ITS '07. 2007 International Telecommunications Symposium*, Fortaleza, Ceara, Brazil, 2006, pp. 207-212.

[8] V. C. Gungor and O. B. Akan, "DST: delay sensitive transport in wireless sensor networks," in *Proceedings of ISCN'06 7th International Symposium on Computer Networks*, Istanbul, Turkey, 2006, pp. 116-122.

[9] at&t, "Text Messaging and Emergency Notification Systems: Important Information for Emergency Communications Professionals," ed. www.business.att.com: at&t, 2008.

[10] F. C. Delicato, L. Fuentes, N. Gamez, and P. F. Pires, "A middleware family for VANETs," in *Ad-Hoc, Mobile and Wireless Networks. 8th International Conference, ADHOC-NOW 2009*, Murcia, Spain, 2009, pp. 379-384.

[11] J. Balasubramanian, D. C. Schmidt, L. Dowdy, and O. Othman, "Evaluating the performance of middleware load balancing strategies," in *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing*, Monterey, CA, USA, 2004, pp. 135-146.

[12] C. Adam and R. Stadler, "Implementation and evaluation of a middleware for self-organizing decentralized Web services," in *Self-Managed Network Systems, and Services. Second IEEE International Workshop, SelfMan 2006.*, Dublin, Ireland, 2006, pp. 1-14.

[13] I. Martin-Escalona and F. Barcelo-Arroyo, "Performance evaluation of middleware for provisioning LBS in cellular networks," in *2007 IEEE International Conference on Communications*, Glasgow, UK, 2008, pp. 5537-5544.

[14] K. Pohl, G. Bockle, and F. v. d. Linden, "Software Product Line Engineering - Foundations, Principales and Techniques," Springer, Ed., ed Berlin, Germany, 2005.

[15] R. Pries, T. Hobfeld, and P. Tran-Gia, "On the suitability of the short message service for emergency warning systems," in *VTC 2006-Spring. 2006 IEEE 63rd Vehicular Technology Conference*, Melbourne, Vic., Australia, 2006, pp. 991-995.

[16] T. K. Group. (2010, *Kannel*. Available: www.kannel.org

[17] *1 Year in Memory of Tsunami: 26 December 2005, Thailand*: Thailand Government, 2005.

[18] "OSI model definition," in *Computer Desktop Encyclopedia*, ed, 2010.

[19] I. F. Akyildiz, M. C. Vuran, and O. B. Akan, "A cross-layer protocol for wireless sensor networks," in *2006 40th Annual Conference on Information Sciences and Systems*, Princeton, NJ, USA, 2006, pp. 1102-1107.

[20] M. Eisenhauer, C. R. Prause, M. Jahn, and M. Jentsch, "Middleware for Wireless Devices and Sensors - Energy Efficiency at Device Level," presented at the 2010 7th Annual IEEE

Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), Boston, MA, USA, 2010.

[21] Y. Faheem, S. Boudjit, and K. Chen, "Data Dissemination Strategies in Mobile Sink Wireless Sensor Networks: A Survey," presented at the 2009 2nd IFIP Wireless Days (WD 2009), Paris, France, 2009.

[22] V. C. Gungor and O. B. Akan, "DST: delay sensitive transport in wireless sensor networks," presented at the Proceedings of ISCN'06 7th International Symposium on Computer Networks, Istanbul, Turkey, 2006.

[23] T. Hasiotis, G. Alyfantis, V. Tsetsos, O. Sekkas, and S. Hadjiefthymiades, "Sensation: A middleware integration platform for pervasive applications in wireless sensor networks," in *2nd European Workshop onWireless Sensor Networks, EWSN 2005*, Istanbul, Turkey, 2005, pp. 366-377.

[24] Y. G. Lyer, S. Gandham, and S. Venkatesan, "STCP: a generic transport layer protocol for wireless sensor networks," presented at the Proceedings. 14th International Conference on Computer Communications and Networks, San Diego, CA, USA, 2005.

[25] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari, "Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks," in *2nd International Conference on Embedded Networked Sensor Systems, SenSys'04*, Baltimore, MA, United states, 2004, pp. 108-121.

[26] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Pump-slowly, fetch-quickly (PSFQ): a reliable transport protocol for sensor networks," *IEEE Journal on selected areas in communications,* vol. 23, pp. 862-872, April 2005.

APPENDIX A – EXPERIMENTS RESULTS

#		SMS	Email	Twitter	Middleware
Experiment 1	Percentage of Success	79.50%	70.73%	70.83%	98.25%
	ACK	318	58	17	393
	NACK, NR	82	24	7	7
	Average Delay (sec)	24.94	99.72	30.17	154.83
	Maximum Delay (sec)	45	170	59	280
Experiment 2	Percentage of Success	77.00%	79.35%	68.42%	98.50%
	ACK	308	73	13	394
	NACK, NR	92	19	6	6
	Average Delay (sec)	24.53	83.83	38.53	146.89
	Maximum Delay (sec)	45	174	59	280
Experiment 3	Percentage of Success	77.00%	83.70%	60.00%	98.50%
	ACK	308	77	9	394
	NACK, NR	92	15	6	6
	Average Delay (sec)	23.71	84.03	34.55	142.29
	Maximum Delay (sec)	45	174	59	280
Experiment 4	Percentage of Success	78.00%	75.00%	68.18%	98.25%
	ACK	312	66	15	393
	NACK, NR	88	22	7	7
	Average Delay (sec)	24.59	84.68	35.4	144.67
	Maximum Delay (sec)	45	172	59	280
Experiment 5	Percentage of Success	76.75%	83.87%	66.67%	98.75%
	ACK	307	78	10	395

	NACK, NR	93	15	5	5
	Average Delay (sec)	24.42	100.71	39.9	165.03
	Maximum Delay (sec)	45	170	60	280
Experiment 6	Percentage of Success	77.50%	75.56%	40.91%	96.75%
	ACK	310	68	9	387
	NACK, NR	90	22	13	13
	Average Delay (sec)	25.17	96.58	29.88	151.63
	Maximum Delay (sec)	45	175	59	280
Experiment 7	Percentage of Success	78.00%	79.55%	61.11%	98.25%
	ACK	312	70	11	393
	NACK, NR	88	18	7	7
	Average Delay (sec)	24.55	99.18	29	152.73
	Maximum Delay (sec)	45	175	60	280
Experiment 8	Percentage of Success	78.00%	80.68%	58.82%	98.25%
	ACK	312	71	10	393
	NACK, NR	88	17	7	7
	Average Delay (sec)	24.88	91.64	40.6	157.12
	Maximum Delay (sec)	45	175	59	280
Experiment 9	Percentage of Success	79.00%	83.33%	35.71%	97.75%
	ACK	316	70	5	391
	NACK, NR	84	14	9	9
	Average Delay (sec)	24.77	96.74	22.6	144.11
	Maximum Delay (sec)	45	175	37	280
Experiment 10	Percentage of Success	79.00%	75.00%	71.43%	98.50%

	ACK	316	63	15	394
	NACK, NR	84	21	6	6
	Average Delay (sec)	23.99	109.36	40.6	173.95
	Maximum Delay (sec)	45	172	60	280
Experiment 11	Percentage of Success	78.75%	75.29%	47.62%	97.25%
	ACK	315	64	10	389
	NACK, NR	85	21	11	11
	Average Delay (sec)	23.97	104.96	26.6	155.53
	Maximum Delay (sec)	45	175	59	280
Experiment 12	Percentage of Success	79.25%	72.29%	73.91%	98.50%
	ACK	317	60	17	394
	NACK, NR	83	23	6	6
	Average Delay (sec)	23.87	94.43	40.35	158.65
	Maximum Delay (sec)	45	172	60	280
Experiment 13	Percentage of Success	78.25%	77.01%	65.00%	98.25%
	ACK	313	67	13	393
	NACK, NR	87	20	7	7
	Average Delay (sec)	24.19	95.52	27.92	147.63
	Maximum Delay (sec)	45	172	53	280
Experiment 14	Percentage of Success	80.25%	81.01%	46.67%	98.00%
	ACK	321	64	7	392
	NACK, NR	79	15	8	8
	Average Delay (sec)	24.04	94.46	36.42	154.92
	Maximum Delay (sec)	45	175	59	280
Experiment 15	Percentage of	79.25%	72.29%	60.87%	97.75%

	Success				
	ACK	317	60	14	391
	NACK, NR	83	23	9	9
	Average Delay (sec)	23.93	99.85	29.14	152.92
	Maximum Delay (sec)	45	175	46	280
Experiment 16	Percentage of Success	75.50%	85.71%	78.57%	99.25%
	ACK	302	84	11	397
	NACK, NR	98	14	3	3
	Average Delay (sec)	24.29	95.38	35.72	155.39
	Maximum Delay (sec)	45	170	60	280
Experiment 17	Percentage of Success	76.75%	78.49%	50.00%	97.50%
	ACK	307	73	10	390
	NACK, NR	93	20	10	10
	Average Delay (sec)	24.54	86.16	38.4	149.1
	Maximum Delay (sec)	44	175	59	280
Experiment 18	Percentage of Success	78.25%	75.86%	66.67%	98.25%
	ACK	313	66	14	393
	NACK, NR	87	21	7	7
	Average Delay (sec)	24.67	92.33	30	147
	Maximum Delay (sec)	45	172	59	280
Experiment 19	Percentage of Success	78.00%	78.41%	73.68%	98.75%
	ACK	312	69	14	395
	NACK, NR	88	19	5	5
	Average Delay (sec)	24.89	103.78	43.71	172.38
	Maximum Delay (sec)	45	172	60	280

Experiment 20	Percentage of Success	77.00%	78.26%	65.00%	98.25%
	ACK	308	72	13	393
	NACK, NR	92	20	7	7
	Average Delay (sec)	23.94	80.2	39	143.14
	Maximum Delay (sec)	45	172	59	280